INTRODUCTION TO PROGRAMMING

Basic Definitions

Programming

A computer as we know consist of the hardware or the tangible parts. We're said to use computer to carry out our day-to-day tasks to process data in order to produce information. The computer itself can not do anything on its own and therefore needs to have instructions which "direct" the hardware in carrying out a given task or tasks. Programming is the technique used to create these instructions. Below are some basic definitions that are relevant to understanding of programming.

Program

This refers a set of instructions that directs the computer hardware in carrying out given task or tasks. For example if you use a computer to compute employee salaries at the end of the month, a program will have been created to do this.

Programming language

This refers to a set of symbols and the rules governing their use that are employed in constructing computer program.

Language translator

This is a program that converts a program written in a language other than machine sensible language into machine sensible language for the computer to execute.

Programmer

This is the person trained in the technique of creating computer instructions (programs).

System

A group elements e.g. people, resources, equipment that work together to achieve a given objective.

Program system

This refers to a group of programs that will work together each carrying its own singular task to achieve the objective of processing information in a certain aspect of an organization.

Types of Programming Languages

In creating computer programs, a media has to be used. In the first computers which were very large, programming involved the "programmer" walking into the computer and physically re-arranging the circuitry in order for a specific task to be achieved. With time however, researchers came up with ways where the programmer could type the instructions through the keyboard.

Machine Languages

These were the first programming languages to be used. The program instructions were written using binary digits (0'sand 1's).

The computer being driven by electricity is a bi-state device in that the internal circuitry is either in "ON" state or "OFF" state.

The 1's are meant to represent "ON" while 0's represent "OFF". When carrying out a given task; the computer internal circuits will keep changing states from "OFF" to "ON" or vise-versa.

When a program was written in machine language, it directly addressed the internal circuit of the machine. It is called so because is the language that the computer "understands".

The language had the advantage of fast program execution since translation was not required, and also the ability of the programmer to directly address and control the internal circuits of the

computer. On the other hand, tracing of errors in program written in machine language was extremely difficult, writing the program itself was time consuming and cumbersome and the programmer had to be expert on the internal working of the computer.

Assembly or low-level languages

These languages replaced machine and were much easier to use. They involve the programmer writing the program instructions using "mnemonic" or symbolic code e.g. using a word like SUB to represent "subtract" or a word like LD to represent "load". Made by computer manufacturers, they were provided with an "Assembler" which is a program used to translate program instruction written in these languages into machine language for execution. They made a task of writing programs much easier for the programmer and were time saving compared to machine languages. On the other hand, one could not use these languages on different type of machine (they were machine specific), they were slower in execution than machine languages and the programmer still had to be an expert in the computers internal working.

High level languages

These were third generation languages and shifted the focus from the machine to the problem, thereby reducing greatly the workload on the programmer. They are still widely in use today, and later developed programming languages are an offshoot of these languages. The languages have the following features:

(i) They have an extensive vocabulary of words, symbols and sentences. Program instructions are therefore written using familiar English statements and mathematical (algebraic) terms.

(ii) A single program statement (instruction) is translated into many machine language instructions.

(iii) Sub-routines and procedures are easily incorporated when writing a program.

(iv) A programmer can work independently of the machine and focus on the problem to solve, high level language are created by software manufacturers and can therefore work on a wide variety of machines (they are portable)

When writing programs using a high level language, the programmer has the advantages of using less item to create the program, ease of error tracing, ability to work on different types of machines and the ability to incorporate" structured programming". Also the programmer does not have to be an expert on the internals working of the machine. On the other hand, some of the languages e.g. scientific languages are hard to learn, they are slow in execution due to the translation process, and the programmer can not effectively address the internal circuitry of the computer as in the previous types of languages.

The other type of language to be discussed later is "object-oriented" languages or "Event-driven" languages.

Language Translators

These are programs that translate programs written in a language other than machine language for execution by the computer. They include the following:-

Assemblers

They translate a source program written in assembly or low-level language into an equivalent machine code object program. A source program is fed through the assembler, which goes through it translating every error free instruction into an equivalent machine code instruction (1:1 ratio). Instructions having errors are not translated, but are put in the source program error listing for the programmer to correct. Only when the entire program is error free the assembler will generate a machine code object program. This is stored for subsequent execution.

Compilers

They translate a source program into a high level language into an equivalent machine code object program. A source program is fed through the compiler, which goes through its translating every error free instruction into many machine code instructions(1: many ratio). Instructions having errors are not translated but are put in the source program error listing for the programmer to correct. Only when the entire program is error free will the compiler generate a machine code object program. This is stored for subsequent execution.

Interpreters

Also translate high level language program instructions to machine code for execution, but in this case, no object program is produced. The programmer, after writing the program, will then attempt to execute it. The interpreter reads each program instruction, translates it and executes it before moving into the next instruction. Where an error is encountered, the circle of "read, translates, executes" is aborted for the programmer to correct the error. After correcting the error on errors, the circle starts again from the beginning of the program. Whenever a program that uses an interpreter is executed, this cycle must take place.

Programming languages that user interpreters are suitable for training beginners since error trapping is made much easier, and execution is usually halted at the exact point where the error has occurred.

NB: The errors which prevent a source program from being translated and executed are referred to as **syntax** or **semantic** errors.

Some errors do not prevent the program from running, but cause production of erroneous results. These are called **logical** errors

The Stages of Program Development

In creating a functional program or program system, a programmer has to follow some steps. These can broadly be classified into three:

1. Determine what is required of the program, i.e. the information that the program or system is required to produce. The programmer must have clear view of what the program or system is meant to achieve.
2. Determining the inputs or data needs if the required output/information from the program or system is to be effectively produced.
3. Determine the processes that will be carried out on the input in order for the program or the system to produce the required or stated output or information. Here, the programmer has to find out exactly what needs to be done on the input data, and in what sequence e.g. storage, retrieval, arithmetic or logical operations etc, in order to come up with the required output or information.

The above are just broad classifications that give an insight into how the programmer approaches a problem in order to produce a working program or system.

The detailed steps to be followed are:

i) Problem analysis

The programmer looks at existing problem or situation for which the program or system is to be created in order to determine exactly what is required. The programmer may communicate with the people in need of the program or systems (these are called the "End-User") in order to determine their information needs from the program or system. They also identify exactly what the program/system is meant to accomplish.

In doing this, the programmer/analyst will find out how the end users operate in their situation, what they do and how they do it, in order to identify the problems that are encountered. This will give the programmer/analyst idea of what to do to overcome these problems.

NB: An area that requires a program or one for which a program is being developed is referred to as "problem area".

ii) Algorithm design

An algorithm design is defined as a logical sequence of steps which are followed to achieve a given task. When the programmer/analyst has identified the information requirements from the program or the system has identified the problems encountered, they will proceed to lay down a series of steps in a sequence that is to be followed in order to solve the problem. These steps will include what is to be input and how, the process to be carried out, and what is to be output.
In laying down the steps to be followed i.e. the algorithm, the programmer will use algorithm development tools e.g. flowcharts, pseudo code, JSP's etc. when developing the algorithm; the programmer should check that the logical arrangement of steps is correct. Lack of doing this will cause a situation where the program or system gives the wrong results.

iii) Program coding

At this stage, the programmer will proceed to translate their algorithm not program instructions using a pre-defined programming language. Each step in algorithm is translated into a corresponding program instruction. At this stage, the programmer should check for errors which might lead to program not working. Spelling errors are an example. The programmer should also write the program in a clear and easily understandable manner, bearing in mind that one time in the future, the program might need to be maintained or modified by someone else. They are therefore required to use appropriate comments and explanations in their programs. Coding is usually done on paper, and in case of some programming languages, special coding sheets are used.

iv) Transcription

After checking that the program they have written is free from errors, the programmer then proceeds to transcribe it. This involves the programmer typing the program into the computer for storage in preparation for compilation. The program written on paper is now typed in through the keyboard. The programmer should be careful at this stage not to make any error which might lead to compilation difficulties.

v) Compilation

This stage involves the source program being translated into its equivalent machine code object program for subsequent storage and execution on the computer. (Refer to the compilers section above to see how compilation works).

vi) Testing and Debugging:

In this stage the program is executed to check whether it is producing the required results and carrying out the required tasks. The programmer uses "Dummy" data (fake data) which is fed into the program. They will usually have manipulated data manually in order to come up with expected results. After the data has been run through the program, the results produced by the program are compared to the expected results. A difference signifies that a logical error exists in the program. If this occurs the programmer will have to go back and modify the source program. In some cases, they may have to go back up to the algorithm development stage to make modification to the sequence of operation and activities. Testing and debugging is important as it checks that the program or system is fully functional and ready for implementation in the problem area.

vii) Documentation and maintenance

This stage involves creating documents for the program/system, and subsequent changes to accommodate new developments or correct errors that were not discovered in the testing and debugging stage.
In documentation, the programmer/analyst is required to produce two sets of documentation:
A **technical manual** detailing the development of the program/system that will help other programmers in subsequent maintenance.

A **user's manual** which will be used for training the end users of the program/system in order for them to get optimum required results from the program/system.

In maintenance, new changes in the organization or problem areas may signify new information need from the program/system, and some times errors do occur in program or system, that required modifications. This is a duty that can be performed by the programmer, and in their absence, another programmer may undertake it guided by the technical manual created during documentation.

NB: some of the above stages may take place concurrently when developing the program or system. For example, when the programmer/analyst carries out the problem analysis, they come up with documentation called the "system specification". This acts as terms of reference when developing the program/system and will form part of the technical manual.
The algorithm (flowcharts etc.) and some source program code also form part of the documentation.

PROGRAM DESIGN TOOLS

Introduction to Algorithms

An algorithm is a sequence of steps which results to a plan or strategy on how to go about in solving a problem. You use a variety of algorithms to solve everyday tasks and you certainly realize that there may be several algorithms or plans that will result in the solution to problem. Thus there are different ways of presenting an algorithm. Some common ways include:

1. Pseudo code
2. Flowcharts
3. Jackson structure diagrams(JSD's)

Pseudo codes

This is a case where an algorithm is expressed in English like statements (description)
For example, the following is a pseudo code for a program to calculate the pay amount for five (5) employees:-

1. Start
2. Initialize counter to 1
3. Enter employee details
4. Compute pay amount
5. Print the pay amount
6. Increment counter by one (counter =counter +1)
7. Check the value of the counter
   If counter < 6
   Loop to step 3
   Else
8. End

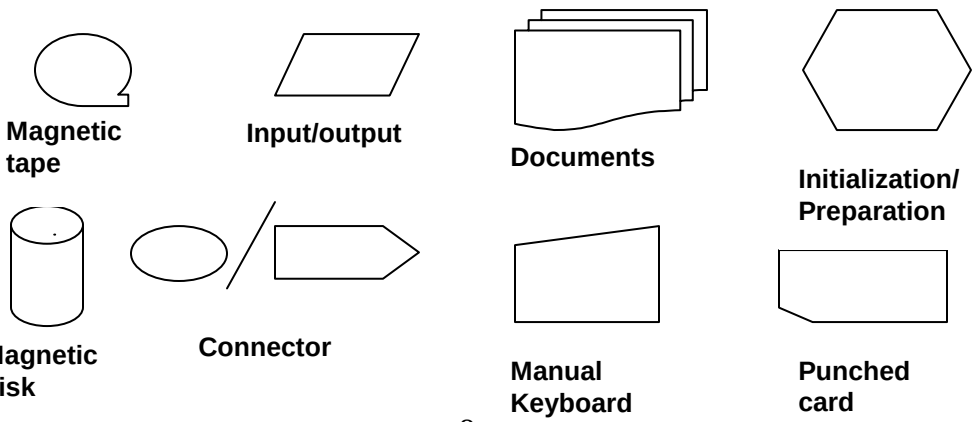
Flowcharts

A flowchart is a representation of a whole program by use of special graphical symbols called flowchart symbols.

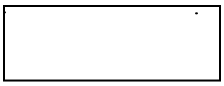
Advantages

1. Provide a visual aid to the programmer
2. Are easier to understand because they are summarized and use standard symbols.
3. One is able to distinguish various stages uniquely by use of symbols
4. They show a sequence of procedures
5. They are good communication aids to both programmers and the users
6. They form part of system documentation

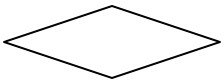
The following are some of the common flowchart symbols:-



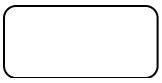
Prepared by:
Ericobanks
Contact: 254700711233
Website: www.codestar.co.ke



Action or process

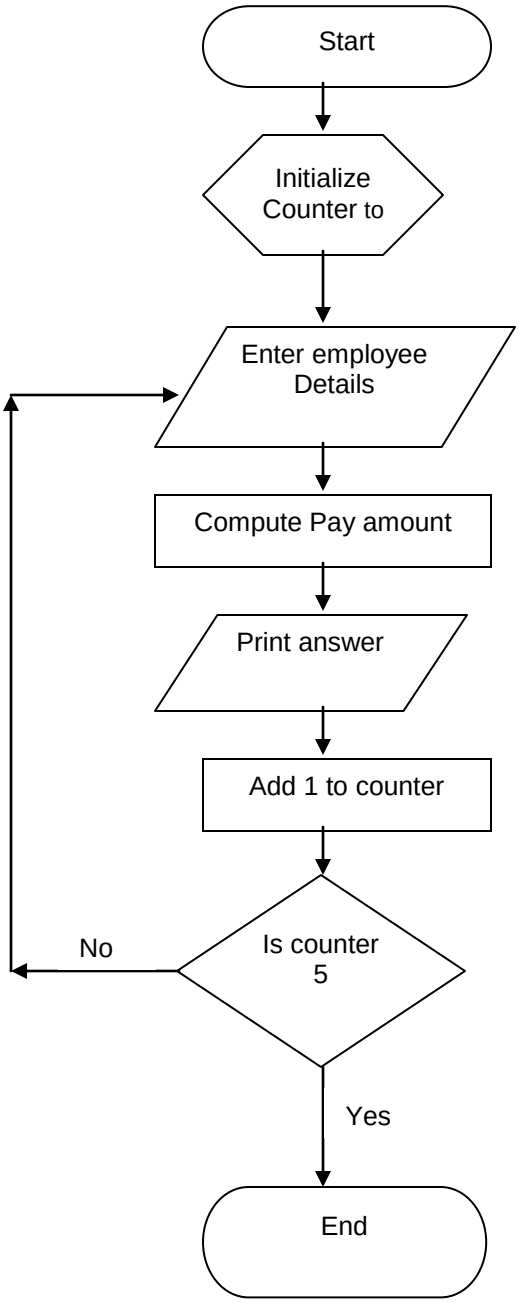


Decision

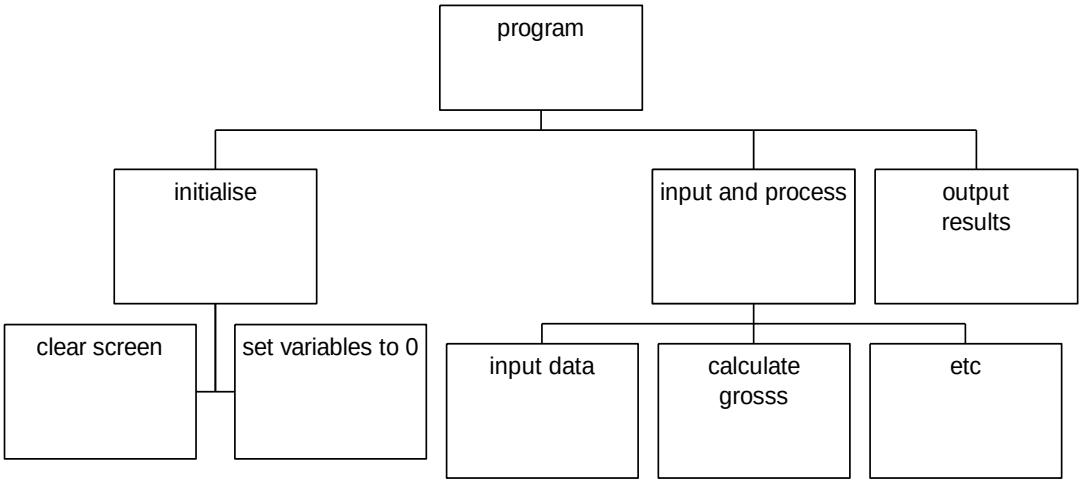


Terminator

The above example, written in pseudo code can be translated into the equivalent flowchart diagram to appear as follows:



Jackson Structure Diagrams
These are block diagrams that are useful in representing the structure of the programs and how the modules are related to each other. They are diagrams that resemble a tree diagram (family tree) with the main program statement written across the top line.



Example
Building blocks
There are three different building blocks for Jackson structure program diagrams. They include:-

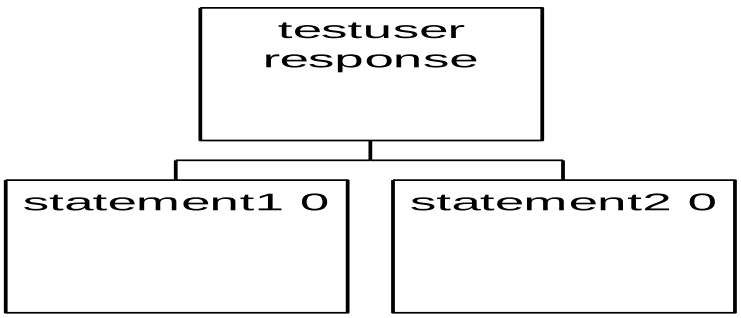
- **Sequence blocks**

One statement follows another and program executes them in the sequence given. In sequential statements plain blocks are used to represent the structure.

- **Selection blocks**

The next statement to be executed depends on the value of an expression. Selection is represented by a small circle in each of boxes representing the alternative paths.
Example:

```
If(answer='y')or(answer='y')
    Statement1
else
   Statement2
end if
```



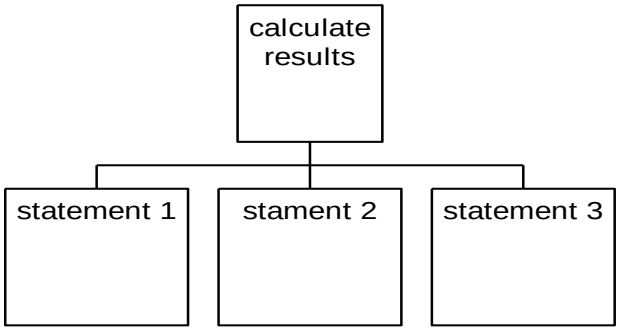
Prepared by:
Ericobanks
Contact: 254700711233
Website: www.codestar.co.ke

- **Iteration blocks**

A section of the program is repeated many times depending on a condition. Asterisk in a box with the condition outside the box and the statement and modules within the loop on the next level down, is used to indicate a section of code repeated many times.

Example:

```
While a<b do
Begin
    Statement 1
    Statement 2
    Statement 3
End {end while}
```



Answer='y'or'y?'

Assignment 1:

As a programmer you are told to write a program for computing the sum of the first 20 positive integers. The program is supposed to output the sum after computing.

a) Write an algorithm for performing the above task using the pseudo code method.
b) Rewrite the algorithm using both the flowchart and the JSP methods.

OBJECT ORIENTED PROGRAMMING

Object Orient Programming (O.O.B) is an approach of creating programs which are window based (Graphical User Interface [G.U.I] applications). It support objects.
Examples of object oriented programming languages are:-

- Visual Basic
- Visual C++
- Visual Fox Pro
- Visual Interter
- Java

Visual Basic (V.B)

It allows the programmer to develop Graphical User Interface (G.U.I) applications. Its event driven programming, meaning that the code remains idle until it is called to respond to some events. e.g. An event like a mouse-click.

Visual Basic is governed by event processor meaning that nothing happens until an event is detected. When the event is detected, the code responds to the event (the event procedure is executed). After execution, the control is returned back to the processor.

There is no pre-determined path the code should follow in even driven application. Since the code can not predict the sequence of events, it makes some certain assumptions about the "state of the world". This means that the commands must be disabled until every field contains a value.

Visual Basic features or capabilities

1. It consists of a full set of objects for drawing an application e.g. Form, command buttons, labels, text boxes etc.
2. It has many icons and pictures for use. E.g. folders.
3. It has response to mouse and keyboard actions.
4. It has clipboard (can copy and paste several times) and printer access facilities.
5. It allows one to write code which instructs the printer to print required information.
6. It has full array of mathematical, string handling and graphical factions.
7. It can handle fixed and dynamic variables and control arrays.
8. It allows sequential and random access file support.
9. It offers useful debugger and error handling facilities.
10. It offers powerful database access tools.
11. It offers packages and deployment wizard that makes distribution of applications or software simple.

Visual Interactive Development Concepts

Visual Basic uses interactive approach to program development. In most languages making a mistake in writing the source code leads to errors being caught by the compiler when compiling starts, the errors are fixed and the compiling process is repeated. Contrary, since Visual Basic supports interactive development, code is interpreted as it is coded (type), catches and highlights most syntax or spelling errors. i.e. It provides "experts watching" facility as the code is being typed a process known as "**catching errors on the fly**".

Visual Basic compiles the code as it is being typed. Running and testing the program takes very short time since much of the compiling is done during typing. Incase of an error, it is fixed and compiling continues without starting again. Due to the interactive nature of Visual Basic, the application is run frequently as it is being developed rather than waiting to compile later.

Visual Basic Objects Concepts.

The objects supported by Visual Basic include forms and controls. Controls are objects that are placed on the form during the interface design. Forms and controls demonstrate the following features.

i) Properties
ii) Methods
iii) Events.

Properties

They are characteristics of an object. Objects of the same type have same properties but different setting or values e.g. Balloon properties include visible attributes such as *height*, *color*, *diameter* etc. A period is used to separate objects from their properties.

Example:

- Balloon.color = Red
- Balloon.diameter = 10

Where Balloon is the Object, color and diameter are the properties, Red and 10 are the values (settings)

Methods

-These are actions that an object can perform or that can be performed on an object. For instance a Balloon has *inflate* method when filled with air and *deflate* method when air is expelled.

Example:

- Balloon.inflate = True
- Balloon.deflate= False

Where inflate & deflate are the actions or methods while True & False are the values

Event

An event is the *response* from the object after an action is undertaken. e.g. a balloon has some predefined actions to certain external events. e.g. a balloon would respond to the event of being punctured by deflating or making noise. In this case puncture is the event.

Example:

```
Balloon_puncture ( )
Balloon.inflate= False
Balloon.deflate= True
Balloon.makenoise= 'Bang'
```

The above code describes the balloon's behavior when a puncture event occurs.

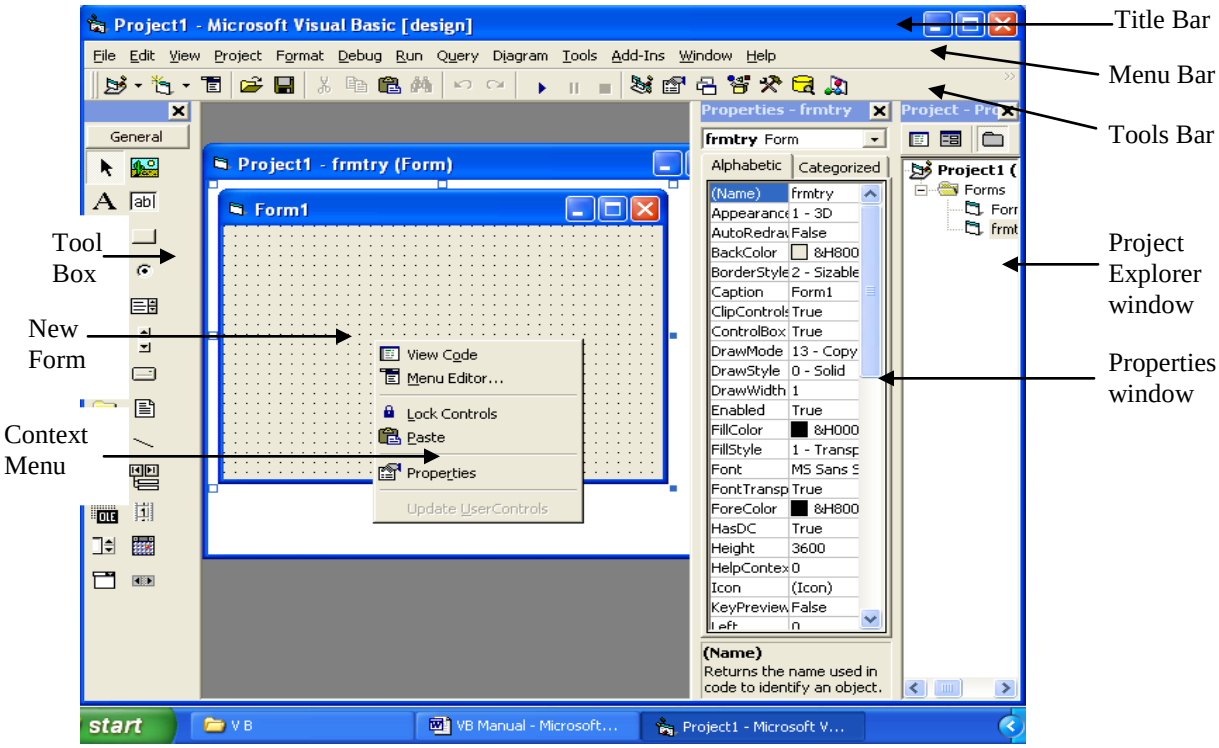
Visual Basic Integrated Development Environment (IDE)

A standard Visual Basic window consists of the following elements.

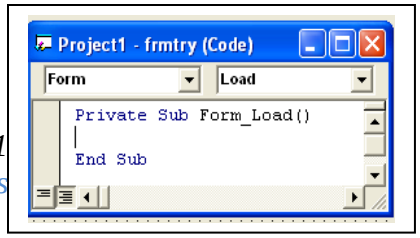
1. ***Title Bar***- Contains the name of the opened project. A project is a Visual Basic file. By default it will be known as 'Project1'.
2. ***Menu Bar***- Appears below the title bar. It has standard menus that contain commands.
3. ***Tool Bar***- Provides quick access to the frequently used menu commands.
4. ***Context menu***- Contains short cuts for the frequently performed actions. It appears when an object is clicked.
5. ***Tool Box***- contains the objects that can be added on the form to create the user interface for the application.
6. ***Form Design window***- This is where the application is designed. On the form, the controls are added to create the G.U.I. It contains a title bar and control box with minimizing, maximizing and closing buttons. User can add or remove forms on the project.

- To add a new form, choose 'Add Form' from 'Project' menu. In the dialog box that appears click on the 'New' tab and choose 'Form'. Click on 'Open' to load the new form.
- To add an existing form from another project, choose 'Add Form' from 'Project' menu. In the dialog box that appears, click on the 'Existing' tab and browse to locate the form to add and click 'Open' to load. It should be of note that forms can not share the same name, so the form to be loaded should not share similar name to any of the forms in the project. The added form is not removed completely from its initial project.
- To remove form, choose 'Remove Form' from 'Project' menu.

Below is a sample of a form design window

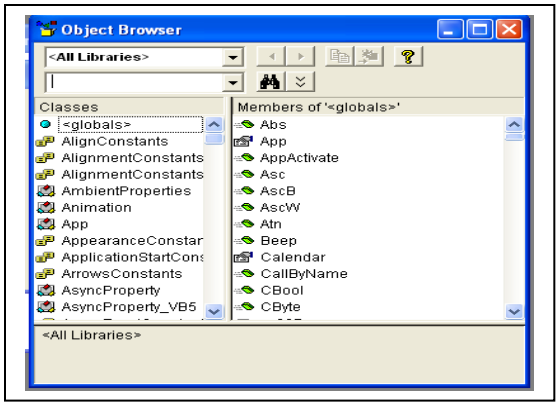


7. ***Project Explorer Window***-It gives the list of collection files used to build the application. It enables one to view the available forms in the project. It allows switching between the object and the view code windows. To access it, choose 'project explores' from 'view' menu.
8. ***Properties Window-*** Gives a list of property setting for the selected form or controls. It describes the characteristics of the object such as size, color etc.To access it, choose 'properties window' from 'view' menu.
9. ***Code Editor Window***-Is where Visual Basic statements for the application are written. The code can be associated with a form or a control in the application. It enables one to view and edit the code. To access the code editor window for any control/ object, double click it and the window will appear.
   Alternatively: Select the control and from 'View' menu choose 'Code'. Any object in Visual Basic consists of two views
   i) *Object view*- Graphical appearance of the object.
   ii) *Code view*- Language statements, constants and declaration that make up the object.



Prepared by:
Ericobanks
Contact: 254700711
Website: www.codes

10. ***Object Browser Window***-Gives the list of objects available for use in the project and a quick way to navigate through the code. It enables one to explore the object in Visual Basic and other applications. To access it, use the 'View' menu and click on 'Object Browser'.



11. ***Form layout window***-It allows viewing the position of the form in the application using a small graphical representation on the screen. It shows the position where the window will be positioned when the application will be running. To access it, use the 'View' menu and click on 'Form Layout Window'.

Debugging Programs

Debugging is the process of identifying and removing errors in a program. The program errors are referred to as bugs.

The program errors may appear in the program due to:

i) Complexity of the program being designed
ii) The masterly skills of the program being designed.

Visual basic language provides several tools which helps to diagnose the errors in a program. It provides debugging tools which makes it easier for the programmer to identify and correct errors in a program.

Types of errors

Programming errors are generally categorized into three groups:

i) Syntax errors
ii) Run-time errors/semantic errors
iii) Logic

Syntax errors

This occurs when the code is constructed incorrectly i.e. when the rules for the particular language are violated.

Examples

- Incorrectly typed keywords
- Omission of required punctuations
- Incorrect construct e.g. failure to terminate 'If' statement with 'End If'.

Visual basic includes an auto syntax check that can detect and correct the syntax error as you write the code. When this option is enabled, visual basic interprets the code as you type it. When it spots an error, visual basic highlights the code and displays a message box explaining the error and offering help.

To set or check the auto syntax option, on tools menu, choose options, then click editor tab on

Run time errors.
This occurs when a statement attempts an operation that is impossible to carry out.
Example:

- Referring to an object that is inaccessible.
- Accessing an undefined variable.
- Attempting to read data from non-existent file.
- Dividing a value with a zero e.g. speed = Distance, if the variable hours is equal to zero (=0) Hours

Then the division is an invalid operation although the formula syntax is correct.

Logic errors
This occurs when the application does not perform the way it is intended to.
These errors are difficult to identify, because the application has syntax valid code, run without performing any invalid operation and still produce incorrect results.
Logic errors are also hard to identify because the language has no way of identifying human intentions.
Examples:

- Use of a wrong operator in arithmetic computations
- Incorrect formula.

To identify logic errors, run the system with a simple data ("dummy" or "test" data), then compare the results with some known results.

Using Debug Toolbar
The programmer can use the extensive debugging tools provides by visual basic to investigate the errors that appears in the program.
Debug tool bar offers quick access to a number of frequently used debugging features. To access the debug tool bar, right click on the toolbar then click debug option and the tool bar appears.

- **Start -** It runs the application from the starter form specified on the project properties dialogue box, if the application is in break mode, the start button changes to continue.
- **End** - It stops the program and returns to design mode.
- **Toggle break point** – It creates and removes a break point. A break point is a place on the code where visual basic automatically halts execution and enters break mode.
- **Step into** – It runs the next executable line of code stepping through each line of the code that follows. If the code calls another procedure, the procedure runs completely before stepping to the next line of the code in first procedure.
- **Local window** – It displays the current values of the local variables.
- **Immediate window** – It displays the immediate window if it is not already displayed. The immediate window allows the programmer to execute the code or query values, while the application is in break mode.
- **Watch window button** – Displays the watch window if it not already displayed. The watch window displays the value of the selected expressions.
- **Quick window** – It displays value of the current expression when the cursor is on the break mode. The expression can easily be added to the watch window.

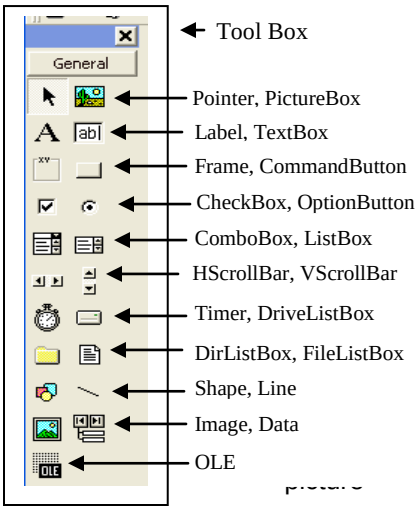
Loading a New Project

A project is a Visual Basic file or document. By default, a new project will have one form. The programmer can add more forms depending on the requirements of the program being designed. Therefore, a project can be compared with a booklet where the forms on the project are taken to be the pages on the booklet.

Steps:

i) Click on the start button. A menu appears.
ii) Point at programs and select "Microsoft visual studio 6.0".
iii) From the menu list that appears click on "Microsoft visual Basic 6.0".The program loads.
iv) In the "New Project" window that appears, click on 'New' tab and double click on "Standard EXE" to open a new form. Alternatively, click on "Standard EXE" and click on "Open" button. A new project is loaded with a single form.

The form is the designing area of the program. With the form on the project, designing of the G.U.I can start by placing the necessary controls on the form. Controls are contained in the toolbox. Therefore controls (objects) are the tools used to design the G.U.I. The following are some of the controls found on the tool box.



Toolbox control **Description**

i) Label - Displays text which the user cannot change.
ii) Textbox - Obtains information from the user or displays information provided by the application.
iii) Command Button- Performs a task when clicked.
iv) Checkbox – Presents a single or several different options to the user. One or all of them can be selected.
v) Option button – Presents a group of options from which the user selects only one option.
vi) Frame – Groups together related controls either visually or functionally.
vii) Picture box – Displays graphic or text. Can be used to animated graphics.
viii) Image – Displays a graphic but uses fewer system than a picture box control. Also supports fewer features than box control.
ix) List Box – Displays a list of items from which the user can select one or several.
x) Data – Used to connect the database with the other controls on the form.

- In designing a program, there are three main steps that are followed. The steps follow one another in this order, Designing the G.U.I, Setting the control properties and Coding (Giving instructions to the controls). After coding, the program is run by choosing 'Start' from 'Run' menu. Alternatively, click on 'Start' tool on the tools bar.

Creating User Interface

The User Interface makes the basis through which the user interacts with the system. This is the first step in designing a Visual Basic program. This is placing the controls where required on the form. To add a control to the form, select it from the toolbox and using the mouse, draw and reposition it where necessary on the form. Alternatively, the control can be double clicked. This will place the control at the centre of the form from where the programmer can position it where necessary.

Setting the Properties

Each object has properties specified to it by default. These are the characteristics of the object. The programmer can change the value of the properties of an object.

Properties are arranged in two ways:-

i) Alphabetic- This is alphabetical order.
ii) Categorized – They are arranged according to various attributes or major characteristics which may need to be set. e.g. Appearance, behavior, data etc.

Caption Property

This is the text/label that appears on the object. It identifies the label for that object. In the caption property, ampersand (&)sign is included which makes it possible to create an ”Access key” which is used to invoke an event using the keyboard rather than the use of the mouse. The key is normally pressed in conjunction with ‘ALT’ key. The ‘&’ sign precedes the letter to be used as the access key.

Example:

Setting on the properties window	Appearance on the control
i) &Save	- Save
ii) E&xit	- Exit
iii) &Ok	- Ok
iv) Can&cel	- Cancel

Note that the preceded by the ampersand sign is underlined on the control. This shows that the character being underlined is the access key. This format is mostly used on command buttons.

Name property

Each control should have a name. The name should be unique in a particular scope. Name is used for identification of the control on the project.

When giving a name to a control, observe the following rules.

i) Start with an alphabetical letter (a - z).
ii) Name consist of either letters only, letter & numbers Or letters, number & underscore. Only these are allowed.
iii) Name should not have a space between words or characters, instead an underscore is used.
iv) Name should not be a Visual Basic keyboard.
v) Name should be unique in the same scope.

Naming conventions

Visual Basic has a naming convention of its objects. The convention includes use of three (3) character-prefixes depending on the type of the object. After the prefix, the other part of the name follows.

Examples:

i) frmstundents – an example of a form
ii) lblstundent - an example of a label
iii) cmdsave - an example of a command button
iv) optmale - an example of an option button

Common objects and their prefixes

Object	Prefix	Example
Check box	chk	chkreadonly
Combo box	cbo	cboenglish
Command button	cmd	cmdcancel
Data	dta	dtabiblio
Directory list box	dir	dirsource

File list box	fil	filsource
Form	frm	frmfileopen
Frame	fra	fralaugnage
Grid	grd	grdprices
Horizontal scrollbar	hsb	hsbvolume
Image	img	imgicon
Label	lbl	lblname
Line	lin	linvertical
List box	lst	lstpolily
Menu	mnu	mnufileopen
OLE	ole	oleobject
Option button	opt	optfrench
Picture box	pic	picdiskspace
Shape	shp	shpcicrle
Text box	txt	txtname
Timer	tmr	tmralerm
Vertical scrollbar	vsb	vsbrate

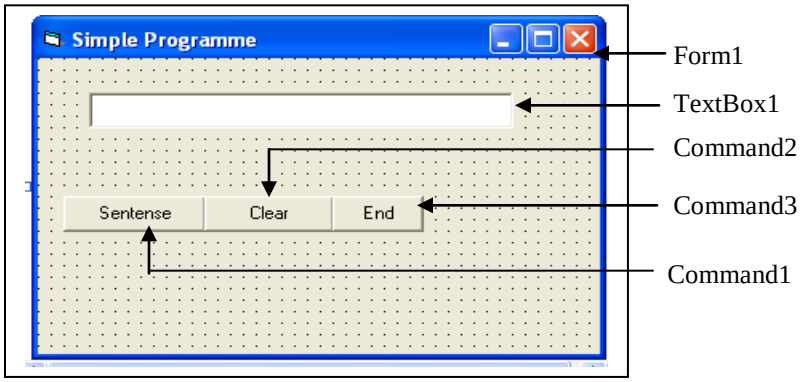
Programming Visual Basic Controls

Text Box and Command Button

Example 1:

Write a simple program that
display the sentence "Welcome to Visual Basic programming" on a textbox after clicking a command button.

Step 1: Design the G.U.I as show below.



Step 2: Set the properties as follows.

Object	Name	Caption
Textbox1	txtsentence	N/A
Command1	cmdsentence	Sentence
Command2	cmdclear	Clear
Command3	cmdend	End
Form1	frmsentence	Simple Program

Step 3: Writing the code

Double click on the command button, cmdsentence and write the following:-

```
Private Sub cmdsentence _Click()        ……1
txtsentence.Text = "Welcome to Visual Basic Programming" ……..2
End Sub        ……..3
```

Double click on the command button, cmdclear and type the following:-

```
Private Sub cmdclear _Click()
txtsentence.Text = ""      ……..4
End Sub
```

Double click on the command button, cmdend and type the following:-

```
Private Sub cmdend_Click()
End      ……..5
End Sub
```

Explanation:

Statements 1 and 3 will always appear by default and enclose the section where the codes should be typed. Statement 1 shows the name of the control (cmdsentence) that contains the instructions (statement 2) to be executed when the event (click) occurs.

The first part of the code (statement 2), txtsentense, is the name given to the text box. After the period, we have 'Text' which is the property of the text box being assigned the sentence. The sentence, "Welcome to Visual Basic programming" is the value of the property 'Text'. Therefore, from this understanding the sentence will appear on the text box after the button is clicked. Remember that the general format is *ObjectName.Property = Value*

Note that the property "text" is for all textboxes but the value can change as shown on the code in statement 4. In this case there in nothing assigned to the property 'Text' and the result after clicking on the button 'cmdclear' will be to clear the contents of the text box.

Statement 5 shows the code which is used to terminate the running of a program. This works similar to the code 'Unload Me' which can be used to replace 'End'.

A single object can have several properties as it is shown on the object's properties window and their values specified in the same coding section as shown below. We can have different properties for the textbox in our example as follows.

```
Private Sub cmdsentence_Click()
txtsentence.Text = "Welcome to Visual Basic Programming"
txtsentence.ForeColor = &H8080FF
txtsentence.Alignment = 2
End Sub
```

Explanation:

The added code specifies other properties (characteristics) of the object txtsentense (textbox) and their values.

Where:

- ForeColor – Specifies a characteristic of text (color) that will be displayed on the text box.
- &H8080FF – Specifies the particular color for that text. This code can be obtained from the properties window by selecting the color and copying the code from the properties window. The copied code is pasted on the coding section.
- Alignment – Specifies another characteristic of text (alignment). The alignment can either be:-
  - Left justify code 0
  - Right justify code 1
  - Centre code 2

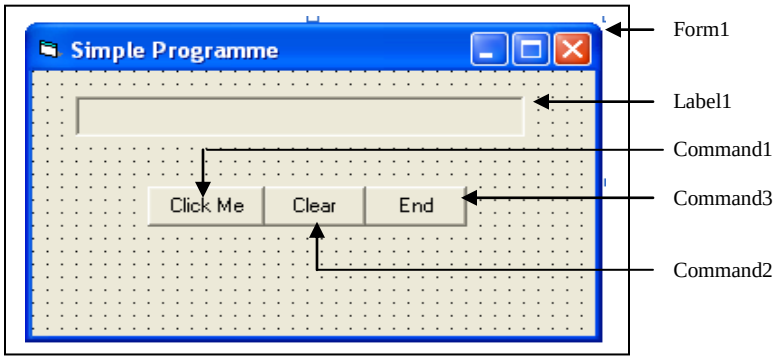
To specify the alignment, specify the code number only. Therefore as indicated on the code, the text on the text box will be placed at the center of the textbox.

Label

Example 2:

In example 1 above, the text displayed on the text box can be changed when the program is running. To avoid this, we can use a label instead of a text box. The text displayed on a label can not be changed by the user of the program when it is running.
To achieve this, we will write another program using a label instead of a text box.

Step 1: Design the same GUI but instead of using the textbox, use a label and give it the name lblsentence. The G.U.I is as shown below.



Step 2: Set the properties as follows;

Object	Name	Caption
Form1	frmlabelsentence	Simple Program
Label1	lblsentense	N/A
Command1	cmdclickme	Click Me
Command2	cmdclear	Clear
Command3	cmdend	End

Step 3: Writing the Code
Double click on cmdckickme command button and type the following.

```
Private Sub cmdclickme_Click()
lblsentense.Caption = "Welcome to Visual Basic Programming"
lblsentense.ForeColor = &H8080FF
lblsentense.Alignment = 2
End Sub
```

Double click on cmdclear on cmdclear command button and type the following

```
Private Sub cmdclear_Click()
lblsentense.Caption = ""
End Sub
```

Explanation:

The two programs will work the same way but the only difference on their working is that for the 2nd one, the text on the label can not be interfered with i.e. no changes can be made.
Label property, similar to textbox 'Text' property is the 'Caption' property. The values of these two properties according to the two programs are the same.

About textboxes and label controls

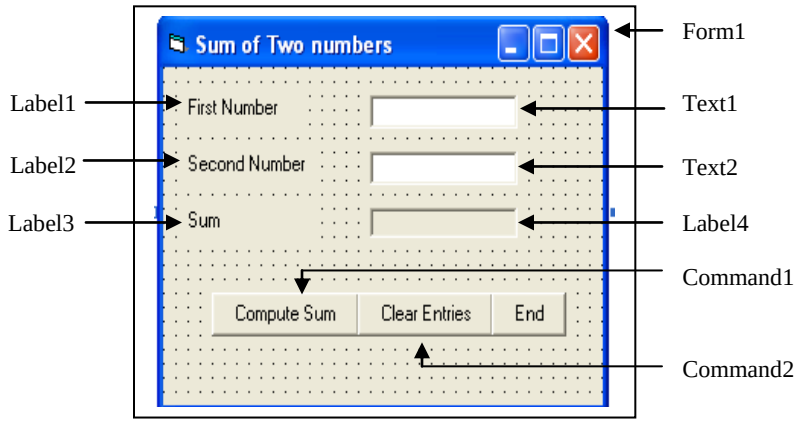
Textbox control is used to display text (data) from a database or indicate text on a code of a particular control as indicated in example1 above. It can also be used to enter data into the form where the user types the data on the text box. The data can be saved into a database.

Since the data in a textbox can be changed especially when the textbox is active. To prevent this we can use a label instead of a textbox. Label control can not be used to get data from the user directly unless the data is input into the label through another control like we shall see as we continue in our study.

The following example will show how we can use text boxes to display text that can not be changed by the user while the program is running.

Example 3:
Write a program that will request for input of two numbers through text boxes and compute their sum after clicking a command button.

Step 1: Designing the G.U.I



Step 2: Setting the properties

Object	Name	Caption	Border style
Form 1	frmsum	Sum of two numbers	
Label 1	lblfirstnumber	First Number	0-None
Label 2	lblsecond number	Second Number	0-None
Label 3	lblsum	Sum	0-None
Command 1	cmdsum	Compute sum	N/A
Command 2	cmdclear	Clear Entries	N/A
Text1	txtfirst	N/A	N/A
Text2	txtsecond	N/A	N/A
Label4	lblanswer	N/A	1 – Fixed single

Step 3: Coding
Double click on cmdsum command button and enter the following

```
Private Sub cmdsum_Click()
lblanswer.Caption = Val(txtfirst.Text) + Val(txtsecond.Text)
End Sub
```

Double click on cmdclear command button and enter the following

```
Private Sub cmdclear_Click()
txtfirst.Text = ""
```

```
txtsecond.Text = ""
lblanswer.Caption = ""
txtfirst.SetFocus
End Sub
```

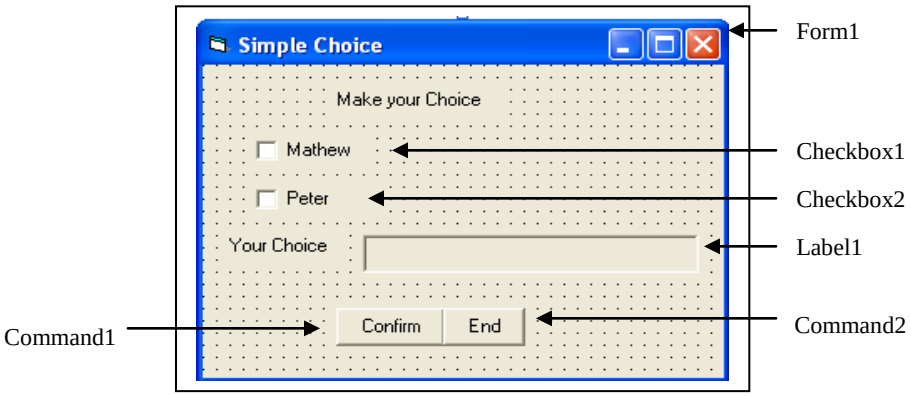
Explanation:
The code on the *cmdsum* button, display the sum of the two values entered into the text boxes on a label. The answer from the computation can not be changed from the label.
The keywords val( ) helps in carrying out the computation of the value entered. It changes the value from string format to numerical format. A string format can not be computed to a mathematical value. Without these keywords, then if 2 and 3 are entered in the text boxes, 23 will be displayed on the label as the answer rather than 5 (the right computational value). Therefore the computation will not be carried out since they will be treated as strings.

The "txtfirst.Setfocus" on the "*cmdclear"* command button is used to put the insertion cursor on the textbox "*txtfirst"* once the command button is clicked. Also this command button clears all the entries on the other controls ie. The text boxes and the label "*lblanswer"*.

Check Box
Check boxes are used to select more than one option from several choices given. If the checkbox is selected, then a tick appears on it and is said to have a value 1 otherwise will have value o.

Example 4;
Design a simple program that will display a message on a label when checkbox is selected or not. The message is displayed after clicking a command button.

Step 1: Design the G.U.I



Step 2: Setting the properties

Object	Name	Caption
Form1	frmchoice	Simple Choice
Label 1	lblchoice	make your choice
Checkbox1	chkmathew	Mathew
Checkbox2	chkpeter	Peter
Command 1	cmdconfirm	Confirm
Command 2	cmdend	End

Step3: Coding
Double click on cmdconfirm and type the following

```
Private Sub cmdconfirm_Click()
If chkmathew.Value = 1 And chkpeter.Value = 1 Then ……1
lblchoice.Caption = "Mathew and Peter"  ……...2
ElseIf chkmathew.Value = 1 And chkpeter.Value = 0 Then  ……..3
lblchoice.Caption = "Mathew"  ……...4
ElseIf chkmathew.Value = 0 And chkpeter.Value = 1 Then  ……..5
lblchoice.Caption = "Peter"  ……6
Else
lblchoice.Caption = "No Choice"  ……7
End If
End Sub
```

Explanation:

Statement 1 means that if both the check boxes are checked (have a tick, value 1) then statement 2 will be executed. Statement 3 means that only the check box for Mathew is ticked but for Peter is not checked. This will executed statement 4. In case there is no check box selected, then statement 7 will be executed.

The selection in execution of statements is as a result of the use of control structures (If-Else If) which will be discussed later in our study.

Option Button

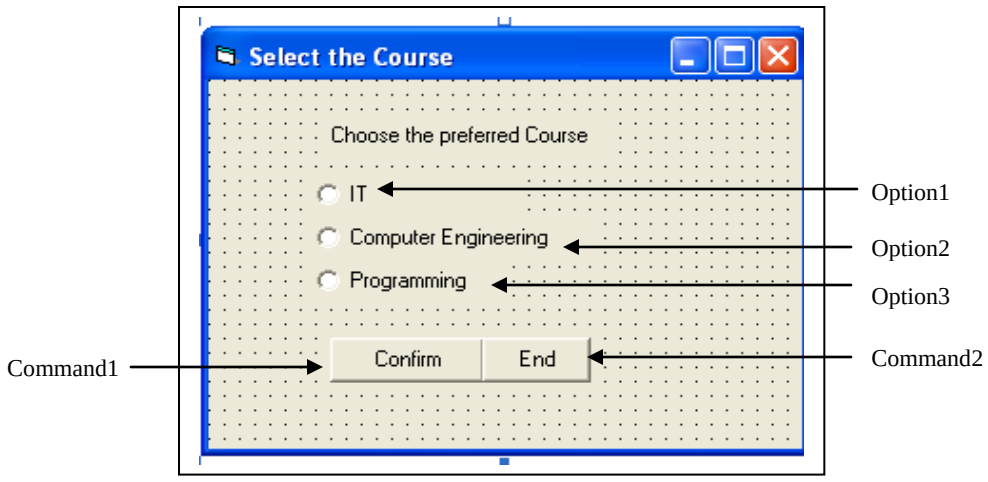
These buttons are used to present a group of options from which the user can select only one option. They are used in areas where some restrictions are required. For instance, where the user has to choose gender (Male or Female). In this case the user if is not careful may choose the two options if a check box is used. To avoid this then option buttons are used instead.

The option button can be turned ON or OFF by the user. When ON, its value is said to be 'True' and when OFF is said to 'False'. An option button will be ON when selected and OFF when not selected.

Example 5:

Write a program that allows one to choose one option from a list of choices and confirm his selection by clicking a command button. The confirmation message should be displayed on a message box.

Step1: Design the G.U.I



Step 2: Set the properties

Object	**Name**	**Caption**
Option 1	optit	IT
Option 2	optengineering	Computer Engineering
Option 3	optprogramming	Programming
Command 1	cmdconfirm	Confirm
Command 2	cmdend	End

Step 3: Coding
Double click on the cmdconfirm and type the following

```
Private Sub cmdconfirm_Click()
If optit.Value = True Then ……1
MsgBox "Information Technology" ……2
ElseIf opteng.Value = True Then ……3
MsgBox "Computer Engineering" ……4
ElseIf optprog.Value = True Then ……5
MsgBox "Computer Programming" ……6
Else
MsgBox "Make a choice", vbCritical ……7
End If ……8
End Sub
```

Double click on the form and type the following

```
Private Sub Form_Load()
optit.Value = False ……9
opteng.Value = False ……10
optprog.Value = False ……11
End Sub
```

Explanation:
Statements 1, 3 and 5 mean that the option buttons are selected. If they are not selected, the value is 'False'. For each case of selection, then the other options are deselected automatically according to the settings of option buttons. Remember that in a particular section on the form, if one option is selected then the other options will be deselected. If 'IT' is selected then statement 1 will be true and statement 2 will be executed. For each case of selection then the corresponding statement is executed. In case that there is no option selected and the command button is clicked, then statements 1, 3 and 5 will be false and will not be executed. Instead statement 7 will be executed.
For statements 9, 10 and 11 put on the form-load event will make sure that all the option buttons are deselected when the form is loading.

Assignment 2:

1. *Write a program that will compute at a time the sum, difference, product and quotient of two integer values entered through textboxes when a command button is clicked. Take care when coding for the quotient not to divide one value by zero. The answer should be displayed on a label.*
2. *Re-write the program above again but don't use the command button to execute the computations but use the option button indicating the computation option to execute the corresponding computation when clicked. The answer should also be displayed on a label.*
3. *Write a program that multiplies three numbers and divides the result with the fourth number. The user inputs the numbers through textboxes and the result is displayed in a label.*
4. *A program is required to compute net salary of employees. The computation is based on the number of hours worked and pay per hour. 10% of the payable amount is given as*

allowance. The employee pays a tax of 16% of the gross pay. The user inputs the hours worked and pay per hour through text boxes. The program should display the following through labels.

- *The amount payable (Hours worked * pay per hour)*
- *Allowance*
- *Tax payable*
- *Gross pay (amount payable + allowance)*
- *Net pay (Gross pay - Tax)*

Combo Box

A combo box is used to hold a list of options where the options are displayed when the drop-down arrow is clicked. After clicking the option of interest from the list, it is displayed on the combo box while the other options are hidden. It displays only one option at a time after selection. There are two ways of adding the contents in a combo box.

i) Through the properties window
ii) Through an operation from a control. e.g. Form-Load or clicking a command button.

I. Through The Properties Window

Steps:

a) Click on the combo box you want to add items and open the properties window.
b) Click on 'List' and type the items one by one. After you type the item press enter and click on the drop down arrow on the list field of the properties window to add the next item. Repeat this until all the items are added on the list.

Combo box doesn't have 'Caption' property but has 'Text' property that holds the text that appears on the combo box when the program is loaded.
You can also set the other necessary properties of interest and then don't forget to name the combo box.

II. Through the Form – load activity or any other control.

After placing the combo box where necessary on the form and setting the necessary properties do the following;-

- **For form – load activity**

Double –click on the form containing the combo box and when the code –editor window opens type the following in the "private sub" section.

Object-name .Additem "Item – name" ……..1

Where:

- Object-name is the name of the combo box you are adding the items.
- Additem is a method (event) used to add items to the combo box.
- Item –name is the item to be added to the combo box.

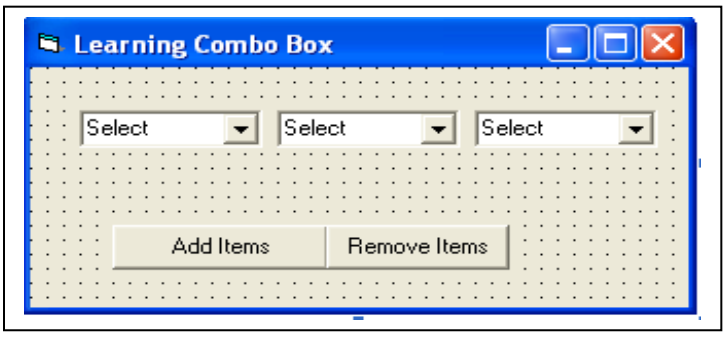
Repeat statement 1 above until all the items to be added to the combo box are exhausted.

- **For any other control like a command button.**

Double click on the control and in the "private sub" section add the commands as shown above on statement 1 (i.e. in the form-load activity)
The only difference between form–load and any other control is only that the coding sections are different. i.e. the form-load you are coding on the form while for any other control you are coding on the control's code section. The other things remain the same.

Example 6:
Design a GUI with three combo boxes and a command button that displays the three concepts discussed above.

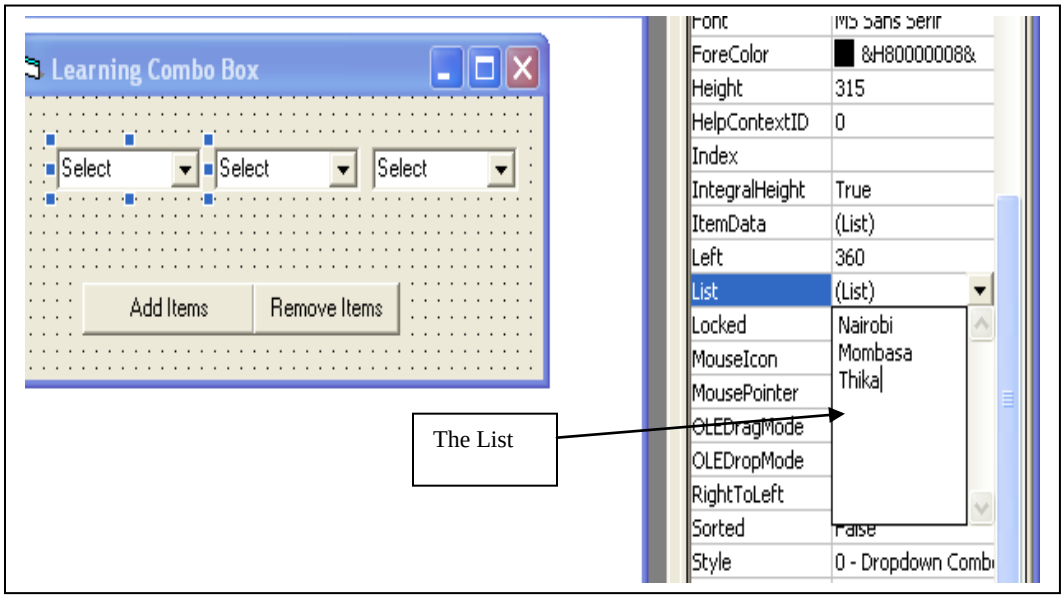
Step1: Design the GUI as follows.



Step 2: Setting the properties

Object	Name	caption	Text
Combo1	cbolist1	N/A	select
Combo2	cbolist2	N/A	select
Combo3	cbolist3	N/A	select
Command1	cmdadd	Add Item	N/A
Command2	cmdremove	Remove Item	N/A

On the properties window for cbolist1 combo box click on the 'List' property and enter the towns one after the other.



The list shown is for the first combo box. This is the first method of populating a combo box. The other two ways are shown on the coding section below.

Step 3: Coding
Double-click on the form and code the following

```
Private Sub Form_Load()
cbolist1.AddItem "Nairobi"
cbolist2.AddItem "Mombasa"
cbolist3.AddItem "Thika"
End Sub
```

Double-click on the command button cmdadd and type the following

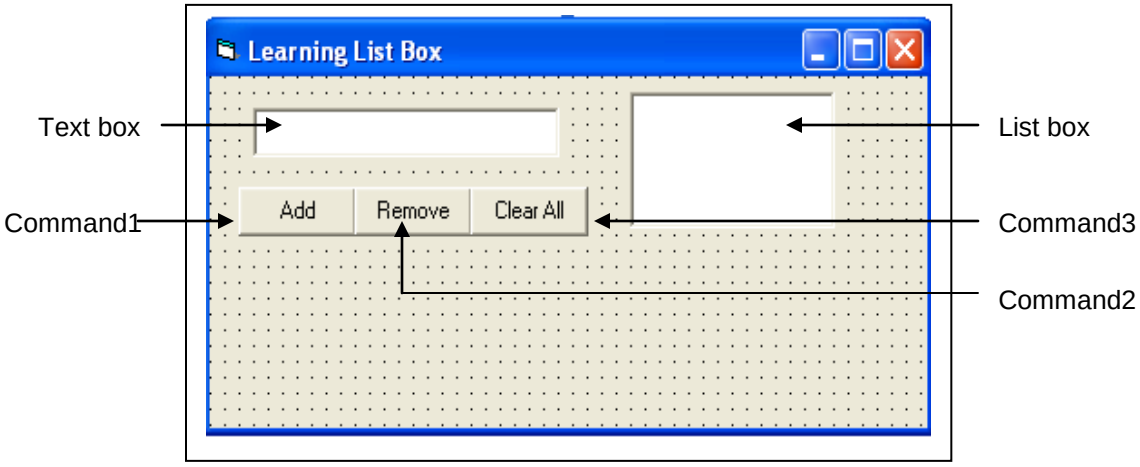
```
Private Sub cmdadd_Click()
cbolist1.AddItem "Nairobi"
cbolist2.AddItem "Mombasa"
cbolist3.AddItem "Thika"
End Sub
```

Explanation:
After running the program, click on the cbolist1 and cbolist2. You will realize that in both, the list appears but if you click on the third combo box (i.e. cbolist3), it is empty until you click on he command button cmdadd to add the elements (items) of the combo box.

List Box
A list box is a scrollable list of choices from which the user can choose one. In visual basic a list is an array of strings formally referred to with the list property (The List property is common to the ListBox and ComboBox). In a list box, one can add items and also remove items. The following example shows how is possible to Add, Remove one-by-one and Remove All the items from the ListBox.

Example 7:
Design a GUI as shown below with a Text box, List box and three Command buttons. One of the command buttons should be adding items to the list box, another command button should be removing the items one-by-one and the last one should be clearing all the contents of the list box.

Step 1: Designing the GUI



Step 2: Setting the properties

Object	Name	Caption
Text1	txtunit	N/A

List1	lstunit	N/A
Command1	cmdadd	Add
Command2	cmdremove	Remove
Command3	cmdclear	Clear

Step 3: Coding
Double click on the cmdadd button and type the following

```
Private Sub cmdadd_Click()
lstunits.AddItem txtunit.Text ……..1
txtunit.Text = "" ……...2
txtunit.SetFocus ……....3
End Sub
```

Double click on the cmdremove and type the following

```
Private Sub cmdremove_Click()
lstunits.RemoveItem (lstunits.ListIndex) ……..4
End Sub
```

Double click on the cmdclear and type the following

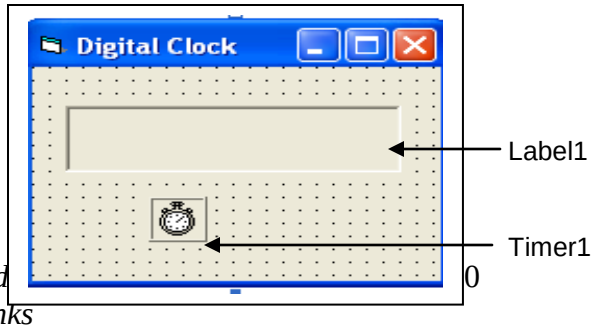
```
Private Sub cmdclear_Click()
lstunits.Clear ……5
End Sub
```

Explanation:
Statement 1 transfers the items from the text box to the list box one-by-one. The key word used to add the text to the list box is 'Additem'. Please note that there is no equal sign between 'Additem' and 'txtunit.Text'. Statement 2 clears the contents of the text box. As the statements will be executed one after the other as they appear on the code, then statement 1 should not be interchanged with 2 because there will be no items to be added to the list box after they are cleared. Statement 3 sets the curser back to the text box.
Statement 4 clears the selected item from the list box. The part that specifies the selected item is the part enclosed in the brackets.
Statement 5 clears all the contents of the list box. The key word used is 'Clear'.

Timer
The timer is one of the windowless controls (controls which do not appear on the form during run time). It is used to automate activities on the form.
The increment of time measured by the timer is set on the properties window. The 'Interval' property is used to set the interval increment. The timer measures time in intervals of 1/1000 of a second. This is milliseconds. Therefore to measure time in one-second interval, the value of the interval should be set to 1000. For half-second interval, the value should be 500.
To automate a given activity, then you have to put the code on the timer's 'Timer' event.

Example 8:
Design a program that will display the system time on a label.

Step 1: Designing the GUI



Step 2: Setting the properties

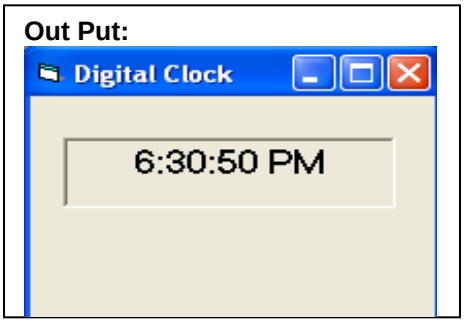
Object	Name	Caption	Interval	Enabled
Label1	lblclock	-	N/A	True
Timer1	tmrclock	N/A	1000	True

Step 3: Coding
Double click on the timer tmrclock and code the following

```
Private Sub tmrclock_Timer()
lblclock.Caption = Time
End Sub
```

Explanation:
The code above will display the system time on the label. The keyword used to pick the system time is 'Time' which is assigned to the caption of the label. Incase you want to display the syatem date you use 'Date' keyword instead of 'Time'. The out put is as shown below.

Out Put:



The timer is not only used to display the system time on a program but it can be used to automate any other event. For instance it can be used to close the program after some defined period of time, move other controls on the form and many more.

Example 9:
Design a program with a close button such that when clicked will fold the form, move it to the left, downwards and then close the program.

Step 1: Design the GUI with a command button to close the program. Put a timer on the form.
Step 2: Name the command button as 'cmdclose' and the timer 'tmrclose'. Set the interval of the timer to be 5 and also disable it.
Step 3: Coding
Double click on the timer and code the following on the 'Timer' event.

```
Private Sub tmrclose_Timer()
If frmtimerclose.Height > 510 Then ……1
frmtimerclose.Height = frmtimerclose.Height – 10 ……..……2
ElseIf frmtimerclose.Width > 1845 Then ……..…..3
frmtimerclose.Width = frmtimerclose.Width – 10 ……..4
ElseIf frmtimerclose.Left < 10000 Then ……..……..5
frmtimerclose.Left = frmtimerclose.Left + 10 ……6
ElseIf frmtimerclose.Top < 7000 Then ……7
frmtimerclose.Top = frmtimerclose.Top + 10 ……8
Else
Unload Me ……..9
```

```
    End If
    End Sub
```

Double click on the close button cmdclose and code the following.

```
    Private Sub cmdclose_Click()
    Tmrclose.Enabled=True          ……10
    End Sub
```

Explanation:

Condition 1 will make sure that statement 2 is executed if and only if the height of the form is greater than 510 pixels. So, the statement 2 will be executed first because the minimum height of the form should be 510. Therefore, statement 2 will be executed and it will reduce the height of the form by 10 pixels after every 5/1000 seconds until when its height will be equal to 510 pixels.

Afterwards, since the width of the form is greater than its minimum width (1845 pixels) then statement 3 will be true hence condition 4 will be executed. This will reduce the width of the form by 10 pixels after every 5/1000 seconds until when the width will be equivalent to 1845 pixels.
'Left' property of the form specifies the position of the form with respect to the left margin of the computer screen. If its position is less than 10000 pixels, then condition 5 is true making statement 6 to be executed. This statement will increase the position of the form from the left screen margin by 10 pixels after every 5/1000 seconds hence making it to move to the right hand side of the screen. This will take place until the position of the form will be equivalent or greater then 10000 pixels, making condition 5 false. If condition 5 is false, then statement 6 will not be executed.

Just like the 'Left' property, the 'Top' property specifies the position of the form with respect to the top margin of the computer screen. If the form's position is less than 70000 pixels, then condition 7 holds true and statement 8 will be executed. This will increase the position of the form from the top margin by 10 pixels after every 5/1000 seconds making it move downwards until its position is greater than or equal to 7000 pixels from the top margin of the screen when condition 7 will be false.

After condition 7 is false, then statement 9 will be executed because all the other conditions so far are false. This statement will close the program.

Therefore, for all these conditions and statements, the form will be folded, moved and later closed. The codes on the timer will be executed when the command button 'cmdclose' is clicked because is the one that will enable the timer (statement 10). Remember that on the settings of the timer, we made it disabled. Therefore, it will remain inactive until activated by clicking the necessary command whereby in our case is the command button.

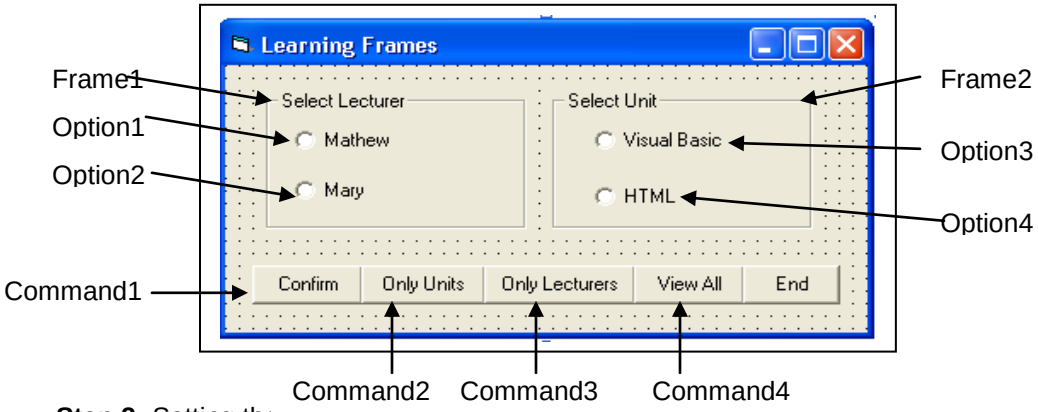
Frames

A frame is used to group controls on the form. They are mostly used to hold sets of option buttons and check boxes. They can also be used to hold other controls. In case of a scenario where many sets of options are to be selected, it's appropriate to place each set of options within a frame as each frame is treated separately. Remember that only one option can be selected from a list of several options placed within the same location. With the use of frames, then it is possible to select more than one option if the option buttons are placed on different frames. Although, the option buttons will be treated separately, this does not mean that the option button on different frames can share the same name. This is because they are in the same form.
When a frame is made visible or invisible, all the controls within it appear or disappear with the frame. In case the frame is made to move, they also move with it.

Example 10:

Design a program that will allow the user to choose a lecturer and the unit that he/she lecturers. Use option buttons.

Step 1: Design the G.U.I as shown



Step 2: Setting the properties

Control	**Name**	**Caption**
Frame1	fralect	Select Lecturer
Frame2	fraunit	Select Unit
Option1	optmathew	Mathew
Option2	optmary	Mary
Option3	optvb	Visual Basic
Option4	opthtml	HTML
Command1	cmdconfirm	Confirm
Command2	cmdunit	Only Units
Command3	cmdlect	Only Lecturers
Command4	cmdall	View All

Step 3: Coding

Double click on the *cmdconfirm* and type the following

```
Private Sub cmdconfirm_Click()
If optmathew.Value = True And optvb.Value = True Then ……1
MsgBox "Visual Basic taught by Mathew" ……2
ElseIf opthtml.Value = True And optmathew.Value = True Then ……3
MsgBox "HTML taught by Mathew" ……4
ElseIf optvb.Value = True And optmary.Value = True Then ……5
MsgBox "Visual Basic taught by Mary" ……6
 ElseIf opthtml.Value = True And optmary.Value = True Then ……7
MsgBox "HTML taught by Mary" ……8
Else
MsgBox "Please meke your choice" ……9
End If
End Sub
```

Double click on the *cmdunit* and type the following

```
Private Sub cmdunit_Click()
fralect.Visible = False ……10
fraunit.Visible = True ……11
End Sub
```

Double click on the *cmdlect* and type the following

```
Private Sub cmdlect_Click()
fralect.Visible = True    ……12
fraunit.Visible = False   ……13
End Sub
```

Double click on the *cmdall* and type the following

```
Private Sub cmdall_Click()
fralect.Visible = True    ……14
fraunit.Visible = True    ……15
End Sub
```

Explanation:

Remember our discussion on option buttons that if the option button is selected, its value is equivalent to 'True'. This means that for statements 2, 4, 6 and 8 to be executed then both the corresponding option buttons should be selected. If you are careful you will realize that the two options to be selected are on different frames. Option buttons on the same frame can be selected one at a time. Therefore the option buttons on different frames are treated to on different sections of the form. Therefore the main purpose of frames is to group similar controls together.

Although the option buttons on different frames are treated differently, this does not mean that they can share the same name. This is because so long as the controls are on the same form then they can not share the same name.

Statement 10 will hide the frame *fralect* and this will make even the option buttons on that frame hidden. This is similar to statement 13 which will hide frame *fraunit.* Statements 14 and 15 will unhide both the frames and their option buttons. This means that frames are used as containers as they contain the option buttons. It is possible to put other controls on the frames.

VARIABLES, CONSTANTS, DATA TYPES AND OPERATORS

Variables

These are memory locations which temporarily store values before execution. They are place holders in the memory for unknown values. On running the program the user provides a value for the variable. For instance calculating the total sales of certain goods, the user provides the values for units sold and the unit price. The value computed is assigned to the variable.

That is: *Total sales = unit price * quantity*

☞ Always before using a variable it must be declared. Declaring is specifying the data type of the value to be held by that variable during program running.

A variable holds different values each time the program is run. Variable enables one to make calculations without having to know in advance what value the user provides.
A variable consists of:-

i. Name – Is the word referring to the value contained in a variable
ii. Data type – Determines the type of data to be held by the variable. e.g. string, integers, data, currency etc

Naming Variables in Visual Basic

There are rules which must be followed when naming variables in Visual Basic. They include:-

i) The first character must be a letter of alphabet
ii) Name can only contain letters, alphanumerical or a combination of the two given and underscores.

Combination	Examples
❖ Letters	myname, num, answer
❖ Alphanumericals	myname1, myname2, netsalary
❖ Letters, alphanumericals with an underscore.	my_Name, My_Name1

iii) Never use a Visual Basic keyword as a variable name.
iv) A variable name should be unique in a particular scope or current procedure/ module.
v) The name should not be longer than 255 characters.

Declaring Variables.

This refers to informing the program in advance that it will encounter such a variable name. Variables are declared using Visual Basic keyword statements depending on their location on the program. For instance, Dim is a keyword which is mostly used.
While declaring the variable name, supply the *variable name* and the *data type*.
When declaring a variable, it is not a must to supply the data type because the variable name can be assigned any value which means that it will have different values.
This is not recommended because;-

i) It can waste memory resources.
ii) Variable type can produce unpredictable default value behaviors, particularly with arrays.

Not specifying the variable type is not advisable to beginners for this may bring confusion during program coding and may also hider one from learning how to use the proper data type for the proper situation.

General syntax will be: Dim variable_ name [As type]
Example;
Dim num **As** Integer
Where: -Dim -Is the keyword used in variable declaration.
- num -Is the variable name
-As - Is another keyword used during declaration

-Integer - is the data type.

☞ A variable declared with the Dim statement exist as long as the procedure is existing. When the procedure finishes, the value of the variable disappears. The variable is local to that procedure.

Other examples.

Dim	firstname	**As**	string
Dim	Totalcost	**As**	Currency
Dim	closedata	**As**	Data
Dim	Age	**As**	Integer

Storing and Retrieving data from variables

Some variables my be declared with predefined values. The assignment operator (=) is used to assign a value to the variable.

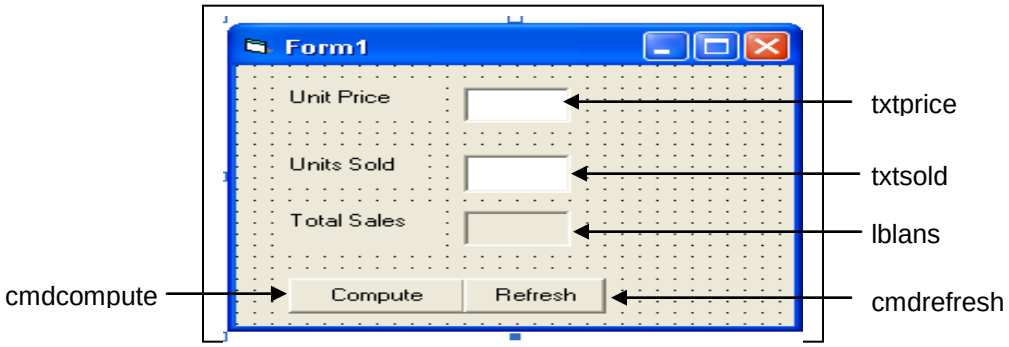
Example:

Dim Age **As** Integer ……Declaring the variable 'Age' to contain Integer values
Age = 20 ……..Assigning 20 to the variable
Dim course **As** String ……Declaring the variable 'course' to contain String values
course = "programming"

Example 11:

Design a program that computes the total sales on items. The user inputs the unit price and the quantity sold through text boxes. The total sales are displayed through a label.

Step 1: Design the G.U.I as shown



Step 2: Coding

Double click on the form (or any control) to open the code window. Open the *General* section and code the following.

Dim unitprice **As Currency**
Dim unitssold **As Integer**
Dim sales **As Currency**

Double click on the command button cmdcompute and code the following.

Private Sub *cmdcompute_Click()*
unitprice = Val(txtprice.Text)
unitssold = Val(txtsold.Text)
*sales = unitprice * unitssold*

```
lblans.Caption = sales
End Sub
```

Double click on the command button cmdrefresh and code the following

```
Private Sub cmdrefresh_Click()
txtprice.Text = ""
txtsold.Text = ""
lblans.Caption = ""
txtprice.SetFocus
End Sub
```

Explanation:

The code on the *General* section of the code window is used for declaration of the variable names *unitprice, unitssold* and *sales.* **Currency** and **Integer** are the data types of the respective variable names. This means that the value to be assigned to the variable names during the program execution will be only of currency and integer types. Since the variables are declared on the general section, then they can be accessed from any part of the program.

In the command button *cmdcompute*, the variable names are assigned different values
i.e. – *unitprice* is assigned the value to be entered in the text box *txtprice*.
-*unitssold* is assigned the value to be entered in the text box *txtsold*.
-*sales* is assigned the product of the two values entered in *txtprice* and *txtsold* text boxes.

The value assigned to *sales* variable name is made to be the caption of the label *lblans*. This means that, the total sales will be displayed on the label.

On the command button *cmdrefresh*, the code clears all the values entered in the text boxes and the product displayed on the label *lblans*. Also, it takes the curser (insertion point) to the text box *txtprice* because of the code **"*txtunitprice.Setfocus*"**

Assignment 3:

1. *A program is required that will compute either the total surface area of a closed cylinder or the volume. The values of the radius and the height will be entered during program execution through text boxes. The answer should be displayed on a label. The program should compute each quantity at a time. Use the necessary controls to take care of this. Design the program.*

Re-design the program above (No.1) but this time round the user can compute both the surface area and the volume at the same time or one of the quantities. The settings remain the same as in question one.

Variable scope:

This defines which part of the code is aware of the variable existence. It refers to the visibility of a variable within a module or application.
There are four variable scopes:-

i) Procedure level/local
ii) Procedure level static
iii) Form / module level
iv) Global level.

i) Procedure level

They are recognized only in the procedure or control in which they are declared. They are local in that they can only be recognized by and in that procedure. They do not retain their value once the procedure terminates. They are good for temporary calculations.
They are declared using the **Dim** keyword. e.g. **Dim** number **As Integer**
The examples above show the procedure level variables.

ii) Procedure level static

They exist the entire time the application is running i.e. they retain their value in between procedures.
They are declared using the **Static** keyword. e.g. **Static** age **As Integer**.

iii) Form / Module level

They retain their value and are available to all procedures within the form. They are declared in the **general section** code window. They are declared using the **Dim** keyword. e.g. **Dim** basicpay **As Currency**.
Private keyword can also be used to replace the **Dim** keyword. e.g. **Private** basicpay **As Currency**.

iv) Global level

These are variables that retain their values and are available to all procedures within the application (project). They should be kept in one module or form. They are declared using the keyword **Global** e.g. **Global** mydate **As Date**.
They are declared at the declaration section at the top of one of the module.

Constants

They are entries that do not change their values during program execution. They have fixed values that do not change.
There are two types of constants in Visual Basic.

i) Intrinsic (system defined) constants

They are provided by Visual Basic application and controls. They are inbuilt in Visual Basic program. They are listed in Visual Basic and for Visual Basic applications in the object libraries.
They start with the keyword **vb** e.g. **vbyes**, **vbRed, vbCancel** etc

ii) Symbolic (user define) constants

They are provided by the user. They are declared using **Const** keyword.
e.g. **Const** NHIF =200

Naming Constants

The rules for naming constant are the same as the rules for naming variables. Constant should be used in uppercase to differentiate them from variable names.

Declaring Constants

The general syntax is as following:-
[public /private] **Const** constantName [As Type] = Expression.
Where:

- [Public/Private] –Defines the scope of the constant
  e.g Public – Indicates that the constant is available for all the applications.
  Private – Indicates that the constant is available for the form in which it is declared.
  - They are called permission levels and are optional.
- Const – is the keyword for defining a const.
- Constant Name – Is valid symbolic name to represent the constant.
- As Type – Define the type of data object for the constant e.g. Integer, string, currency etc.
- Expression – Is the value to be held by the constant. It can be a numerical value, a string, a symbolic constant or an expression

Examples:

1. Private const MAXPLANETS AS Integer =9

2. Const PI = 3.14
3. Const SEC_ PER_HOUR =60*60

1. To define a string constant, the value must be enclosed in double quotes.
   e.g. const COUNTRY = “Kenya”

2. It’s possible to place more than one constant declaration in a single line by separating them with a comma. e.g. const PI = 3.142, ZIPCODE = “254” ; In this case, the keyword **Const** is used only once.
3. The expression on the right side of the assignment operator can be a number, a string or an expression that result to a number or another defined constant.
   e.g. PI2 = PI *2 – expression. The constant can only be placed on the code once it has been declared.

Data Types

You can store almost anything in a variable. A variable can hold a number, a string of text, or an instance of an object including form, controls and database objects. It can hold any type of information, but different types of variables are designed to work efficiently with different types of information.
Different data types have different memory requirements and can hold different range of values. At some point you may wonder whether you are using a variable of the right size, particularly with regard to numbers. But a good rule of thumb is to go with the larger variable size if you don’t have a very good idea of what the limit of your variable will be.
e.g. If you think that your program at some point might be required to use number with decimals or fractions, you might want to use double and single variable type instead of Integer and Longs

Type	**Stores**	**Memory Requirements**	**Range of values**
Integer	Whole Numbers	2 Bytes	-32,768 to 32,767
Long	Whole Numbers	4 bytes	Approx. +/- $21.x10^{9}$
Single	Decimal Numbers	4 Bytes	$-3.402823x10^{38}$ to $-1.401298x10^{-45}$ for negative values and $1.401298x10^{-48}$ to $3.402823x10^{38}$ for positive values
Double	Decimal Numbers(double precision floating point)	8 Bytes	$-1.79769313486232x10^{308}$ to $-4.9406564581247x10^{-324}$ For negative values and $4.9406545841247x10^{-324}$ to $1.797691348623x10^{308}$ For positive values.
Currency	Numbers with up o 15 digits left & 4 digits right of the decimal.	8 Bytes	922,337,203,685,477.5808 to 922,337,203,685,477.5807
String	Text information	1 byte per character	Up to 6500 characters for fixed length strings and up to 2 billion character for dynamic strings.
Bytes	Whole Numbers	1 byte	0 to 255
Boolean	Logical values	2 bytes	True or False
Date	Date & time information	8 bytes	Jan 1st 100 to Dec 31st 9999

Object	Pictures & any object reference	4 bytes	N/A
Variant	Any of the preceding data types	16 bytes + 1 byte per character	N/A

Visual Basic Operators

Mathematical Operators

They are used to carry out computations in Visual Basic. Visual Basic supports a number of different mathematical operators that can be used in program statements.
The table below summaries the mathematical operators and their corresponding Visual Basic symbols that can be used when programming.

Operator	Symbol
Addition	+
Subtraction	-
Multiplication	X
Division	/
Integer division	\
Modulus	Mod
Exponentiation	^

❖ Addition, Subtraction, and Multiplication Operators

These are the simplest. They are put between the operands on the right side of the assignment operator (=)

e.g. sum = num1+ num2
diff = total - deduction
prod = sal * days

Where – *num1, num2, total, deductions, sal* and *days* are the operators to be computed (operated)
- *sum, diff* and *prod* are the variables or object properties that are assigned the computed values after the computation.

❖ Division Operator

This operator is a little bit complicated than multiplication operator. This is because there are 3 types as shown on the table above.
For the division where the forward slash **(/)** is used called the **floating – point – division** (normal type of division). This type of division returns a number with its decimal portion if it has.

e.g. sum = 9/4
sum = 2.25

For the division where the backward slash **(\)** is used is called the **Integer division**. This returns the whole number only incase the two values involved would give a quotient with a decimal/fractional part. The decimal/fractional part is truncated.

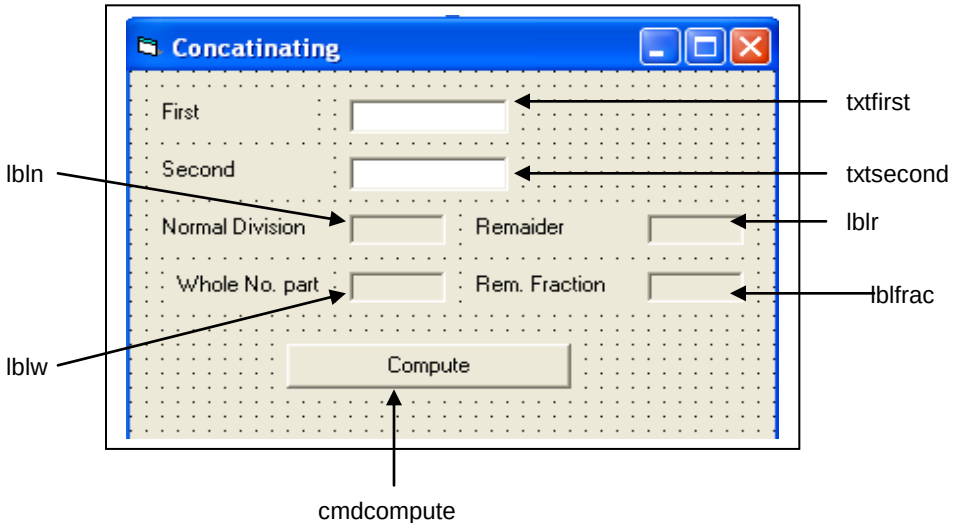
e.g. sum = 9\4
sum = 2

For modulus division where **Mod** keyword is used, the remainder (which results to the decimal/fractional part of a quotient) is given but the whole number is truncated.

e.g. x = 9 **Mod** 4 gives 1
y = 15 **Mod** 4 gives 3

Example 12:
Design a program that will employ the three types of division operator.

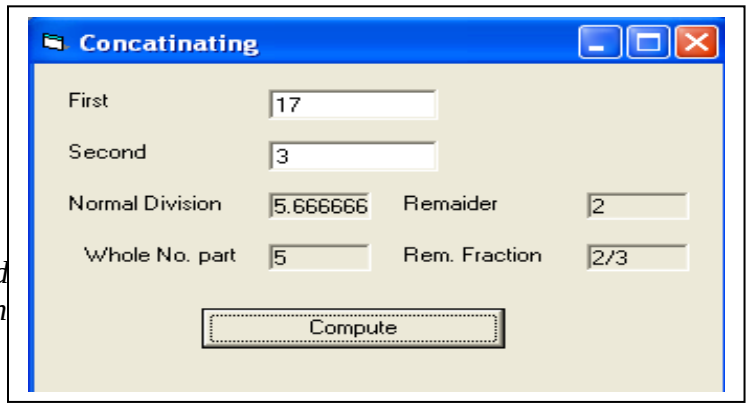
Step 1: Design the G.U.I. as shown below



Step 2: Coding
Double click on the coming button and type the following

```
Private Sub cmdcompute_Click()
lbln.Caption = Val(txtfirst.Text) / Val(txtsecond.Text)        ……..1
lblw.Caption = Val(txtfirst.Text) \ Val(txtsecond.Text)        ……..2
lblr.Caption = Val(txtfirst.Text) Mod Val(txtsecond.Text)      ……..3
lblfrac.Caption = lblr.Caption & "/" & txtsecond.Text          ……..4
End Sub
```

Explanation:
Statement 1 computes the normal division (Floating-point-division). It returns values with decimal values. Statement 2 computes the integer division. It returns the whole part of the quotient only. Statement 3 computes the remainder of the quotient only. It returns the remainder only. Statement 4 will be used to display the fractional part of the quotient as it should be on normal mathematical computation. It will take the remainder displayed on the label *lblr* and display it as the numerator of the value entered on the text box *txtsecond*. The '&' symbols are used to concatenate (join) Visual Basic statements. Therefore in this case they are used to join the text on the label *lblr,* the forward slash (/) and the text on the text box *txtsecond.*
If we run the program we can have an out put as shown below.



Other Visual Basic Operators
The other Visual Basic operators include **concatenation (string) operator, comparison operators** and **Logical operators.**

Order of precedence in statements
In Visual Basic, the order of execution of operators depends on their precedence. Operators with higher precedence than others are executed first if they are in a single statement.
Mathematical operators are performed first, then comparison operators and finally logical operators.
The table below shows the order of Visual Basic operators

Operator	Operator Symbol
Exponential	^
Negation (unary)	-
Multiplication, Division	*, /
Integer division	\
Modulus	Mod
Addition, subtraction	+, -
Concatenation (string)	&
Comparison operators	=, <>, <, >, <=, >=
Logical operators	Not, And, Or, Xor, Eqv, Imp, Like, Is

Experienced programmers use parentheses to define the ordering of operations within their programs. Using parentheses to define order of precedence removes ambiguity. The use of parentheses does not affect the speed of your program and assures the accuracy and clarity of your code.

Control Structures
They determine the order of execution of program statements. By default, the program is executed from left to right and from top to bottom. This is called the **sequential** control structure. This is related to the way one writes down on a paper. i.e. from the left side of a line to the right side and from the first line at the top of the page to the last bottom line of the page.
The programmer can control the way the statements will be executed in the program. Inclusive of the default control structure we have three different types of control structure;

i.) Sequential control structure
ii.) Selection / Decision control structure.
iii.) Iteration / loop control structure.

Sequential
In this category of control structure we don't have much to discuss because is the default way of statement execution.
Example.

```
Dim    Age  As    integer …….. 1
 Dim    money  As currency …… 2
```

In this example, line 1 will be executed first and then line 2. In line 1, the keyword 'Dim' will be executed first, then Age and the order will continue till the last word in that line.

Decision structure

The statements are executed conditionally. A logical test is carried out using relational operators and the Visual Basic procedures perform different operations depending on the outcome of the test. The decision structure allows choosing a statement to execute out of a given alternatives. It normally requires comparison of magnitude of quantities.

Some of the relational operation include:-

Greater than	>
Less than	<
Greater than or equal to	> =
Less than or equal to	< =

There are two types of decision structures in Visual Basic

a. If statement
b. Select case

A. <u>If Control structure</u>

It allows an application written in Visual Basic to respond to different situations depending upon the result of a logical test. There are 3 ways of using 'if' statement.

i) If – Then – End If statement

It is used where the program executes the statement only if the condition is true. If the condition is false, the statement is skipped. It does not provide an alternative for what will happen if the condition is false.

General syntax:

If *condition* **Then**
Statement
End If

Examples:

i) **If** age > =20 **Then**
MsgBox "Adult"
End If

ii) **If** marks > = 60 **Then**
remarks = "Pass"
End If

The 'If' constant can be written in two ways:-

❖ **<u>Single line syntax</u>**

General syntax:

If *condition* **Then** *statement*

e.g **If** Age > = 20 **Then** MsgBox "Adult"

In this format the *statement* is written in the same line with the *condition*. This syntax does not require "**End If**" as the terminator of the "If" construct.

❖ **<u>Block / multiple line syntax</u>**

General syntax:

If *condition* **Then**
Statement
End If

In this syntax, the If construct must be terminated with "End If" terminator. The *condition* is put on a different line with the *statement*.

ii) If – Then – Else statement

It allows defining two blocks of statements whereby only one is executed depending on the outcome of the test. It evaluates to True or False.

The General syntax:

If *condition* **Then**
Statement1
Else
Statement 2
End If

If the condition is true, then statement 1 is executed otherwise statement 2 is executed.

e.g. **If** age > = 20 **Then**
MsgBox "Adult"
Else
MsgBox "Child"
End If

If the value assigned to '*age'* is greater or equal to 20, then it will give "*Adult*", otherwise *"child".*

iii) If – Then – Else If statement

If allows to define more than two blocks of codes of which it is only one block which will be executed depending on the outcome of the test.

General syntax:

If *condition 1* **Then**
Statement 1
ElseIf *condition 2* **Then**
Statement 2
.
.
.
ElseIf *condition n* **Then**
Statement n
Else
Statement x
End If

Explanation:

If *condition 1* is true, *statement 1* will be executed and if *condition 2* is true, *statement 2* will be executed. This continues by testing the conditions until *condition n*. Then condition that holds true, its corresponding statement will be executed. Incase there is no condition holding true, then *statement x* will be executed.

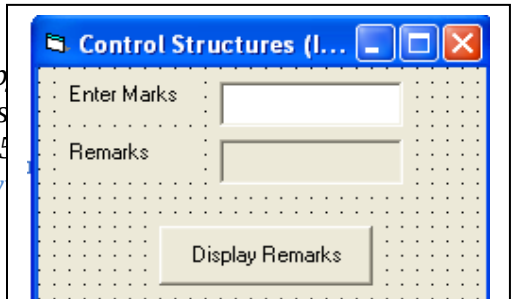
Example 13:

A program is required to allocate remarks to marks obtained by students as follows:-

70% to 100% - Distinction
60% to 69% - Credit
50% to 59% - Pass
0% to 49% - Fail
Any other -Invalid

The marks should be entered through a test box named *txtmarks* and the remarks displayed through a label named *lblremarks* when a command button named *cmdcomp* is clicked.

Step 1: Design the GUI as shown below



Step 2: Coding
Double click on the command button *cmdcomp* and code the following codes.

```
Private Sub cmdcomp_Click()
If Val(txtmarks.Text) >= 70 And Val(txtmarks.Text) <= 100 Then .....1
lblremarks.Caption = "Distinction"
ElseIf Val(txtmarks.Text) >= 60 And Val(txtmarks.Text) < 70 Then .....2
lblremarks.Caption = "Credit"
ElseIf Val(txtmarks.Text) >= 50 And Val(txtmarks.Text) < 60 Then .....3
lblremarks.Caption = "Pass"
ElseIf Val(txtmarks.Text) >= 0 And Val(txtmarks.Text) < 50 Then .....4
lblremarks.Caption = "Fail"
Else
lblremarks.Caption = "Invalid"  ……..5
End If
End Sub
```

Explanation:
Statements 1, 2, 3 and *4* will take care of the different ranges given. Incase he user inputs any entry and it lies in any of the given ranges, the corresponding statement will executed. If the marks entered are above 100 or less than 0, then *statement 5* will be executed.

Assignment 4:
A program is required to compute discount on the amount of goods sold. The amount of discount depends on the quantity of sales made. The sales amount categories include:-

20,000 and above	*-16%*
15,000 to 19,999	*-14%*
10,000 to 9,999	*-12%*
3,000 to 9,999	*-10%*
5,000 to 7,999	*-6%*
0 to 4,999	*-No discount*
Any other entry	*-Invalid Entry.*

The amount of sales should be input through a text box and the computed discount displayed through a label when a command button is clicked. Design the GUI, set the properties of the required controls and write complete code for the program.

B. Select – Case Control structure
It allows for multiple choice of selection of items at one level of a condition. This is far much neater way of writing multiple 'If' statements.
The general format is:-

Select Case (*text expression*)
(**Case** *expression list 1*)
(*Statement block 1*)
(**Case** *expression list 2*)
(*Statement block 2)*
.
.
.
Case Else
(*Statement block n*)
End select

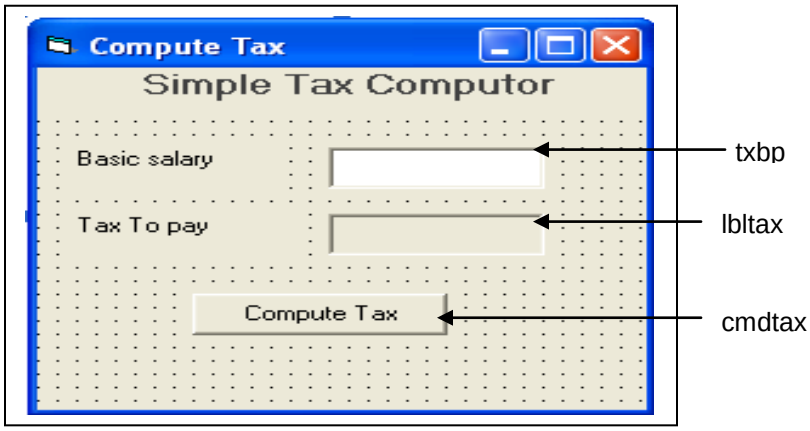
The statement to be executed will be determined by the outcome of the case expression. If the case expression executes to true then the corresponding statement will be executed.

Example 14:

A program is required to compute tax on basic salary figures depending on the ranges indicated below:

50,000 and 49,999	-16%
46,000 to 49,999	-14%
30,000 to 39,999	-12%
20,000 to 29,999	-10%
Less than 20,000	- No tax.

Step1: Design the GUI as shown below



Step 2: Coding

```
Private Sub cmdcomp_Click()
Dim bp As Double
Dim tax As Double
bp = Val(txtbp.Text)
Select Case bp
Case Is >= 50000
tax = (16 / 100) * bp
lbltax.Caption = tax
Case Is >= 40000
tax = (14 / 100) * bp
lbltax.Caption = tax
Case Is >= 30000
```

```
tax = (12 / 100) * bp
lbltax.Caption = tax
Case Is >= 20000
tax = (10 / 100) * bp
lbltax.Caption = tax
Case Else
lbltax.Caption = "No Tax"
End Select
End Sub
```

Iteration / Loop control structure

They make the operation to be repeated many times depending on the value of the condition. Loops are blocks of code that are executed repeatedly. They typically require the *initialization* of one or more variable, *a test* to determine if the loop should continue or stop, and *a counter variable* that changes as the loop executes in order to bring about the fulfillment of the stopping condition.

There are 3 main types of these control structures.

i.) For – Next loop
ii.) Do Loops
iii.) While – Wend

A. For – Next loop

This type of loop is best used when the number of loops is known.

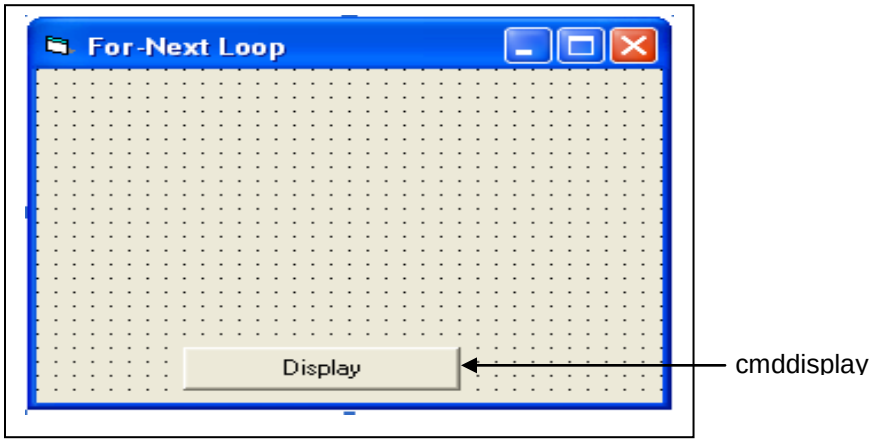
The general syntax is;

For *counter* = *start* **To** *end* [stop increment]
Statements
Next [counter]

Example 15:

Design a program to display numbers 1 to 10 on the form.

Step 1: Design the GUI as shown

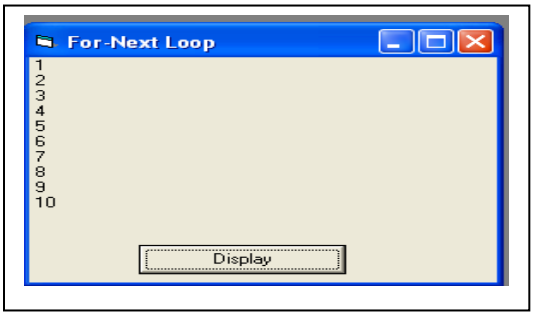


Step 2: Coding

Double click on the command button *cmddisplay* and code the following

```
Private Sub cmddisplay_Click()
For num = 1 To 10 Step 1        .....1
Print num                       ……2
Next num                        ……3
End Sub
```

Explanation:
Statement 1 specifies the initial point (1) and the terminal point (10). Also it specifies the steps (1 step) from one number to the other. Statement 2 prints the numbers on the form. Statement 3 is the incremental operator which will increase the counting from one point to the other.
The out put will be as shown:

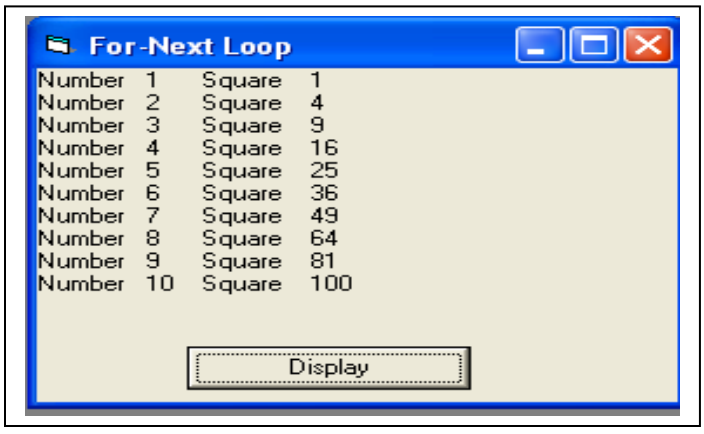


Example 16:
Design a program to display numbers from1 to 10 and their squares.

Step 1: Designing the GUI as shown on example 15 above.
Step 2: Coding
Double click on the command button and type the following code

Private Sub *cmddisplay_Click()*
;

Explanation:
On statement 1 above the underlined part should be double quoted. This part will be displayed as it appears on the code and is joined with the 'known' parts of the code (num) by concatenating (&) operator. This operator is used to join strings.
The out put will be as shown below.



Assignment 5:
Modify the program above to make it more user friendly by allowing the user to be specifying the lower limit, upper limit and the power of computation of the numbers displayed.

B. Do – Loops

They are used to execute block of statements as long as the condition is true. They offer the most flexible way to repeat set of lines.
They are of two types.

i) Do until (while)
ii) The Do

i Do While

This loop has the general format as follows;-

Do Until or **While** [*condition*]
Statements
…..[*Exit Do*]..
Loop

When Visual Basic executes this Do loop, it first test the condition. If the condition is false (zero) it skips past all the statements. If its true (non zero), Visual Basic executes the statements and then goes back to the Do while statement and tests the condition again. Thus the statements never execute if the condition is initially false.

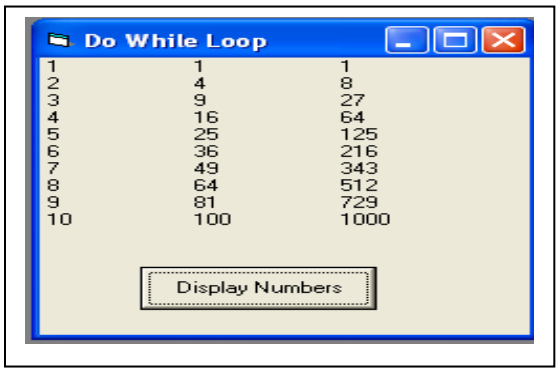
Example 17:

The following code on a command button named *cmddisplay* will display numbers from 1 to 10.

```
Private Sub cmddisplay_Click()
num = 1                    ……1
Do Until num = 11          ……2
Print num, num * num, num ^ 3  ……3
num = num + 1              ……4
Loop                       ……5
End Sub
```

Explanation:

Statement 1 is used to initialize the loop and 2 the terminating value. Statement 3 is for displaying the numbers, squares and the cubes. The three computations should be separated with commas. Statement 4 specifies the steps from one number to the other. In this case the step is 1 (meaning that a number minus its previous number gives 1). Statement 5 makes the program loop.
The output will be as shown:



ii The Do loop

Do-Loop statement execute the statement first and then tests condition after each execution of statements.
The general format is:

Do
Statement
[*Exit Do*]
Loop until or **white** *test*.


The two forms (Do while and Do loop) are effectively the same, except that the logic of a while test is reversed.

Example 18:
The following code on a command button named *cmddisplay* will display numbers from 1 to 10. It uses 'Do-Until' loop.

```
Private Sub cmddisplay_Click()
num = 1
Do
Print num, num * num, num ^ 3
num = num + 1
Loop Until num = 11
End Sub
```

Example 19:
The following code on a command button named *cmddisplay* will display numbers from 1 to 10. It uses 'Do-While' loop.

```
Private Sub cmddisplay_Click()
num = 1
Do
Print num, num * num, num ^ 3
num = num + 1
Loop While num <= 10
End Sub
```

C. While-Wend Loop

This loop executes a block of statements while a condition is True.
The general syntax is:

```
While condition
Statement-block
Wend
```

If the *condition* is true, all the statements are executed and when the 'Wend' statement is reached, control is returned to the 'While' statement which evaluates the *condition* again. If the *condition* is still True, the process is repeated. If *condition* is False, the program resumes with the statement following the 'Wend' statement.

Example 20:
The following code prompts the user for numeric data. The user can type a negative value to indicate that all values are entered.

```
Private Sub cmddisplay_Click()
num = 0
While num >= 0
total = total + num
num = InputBox("Please enter another number")
Print num          ……1
Wend
End Sub
```

Explanation:

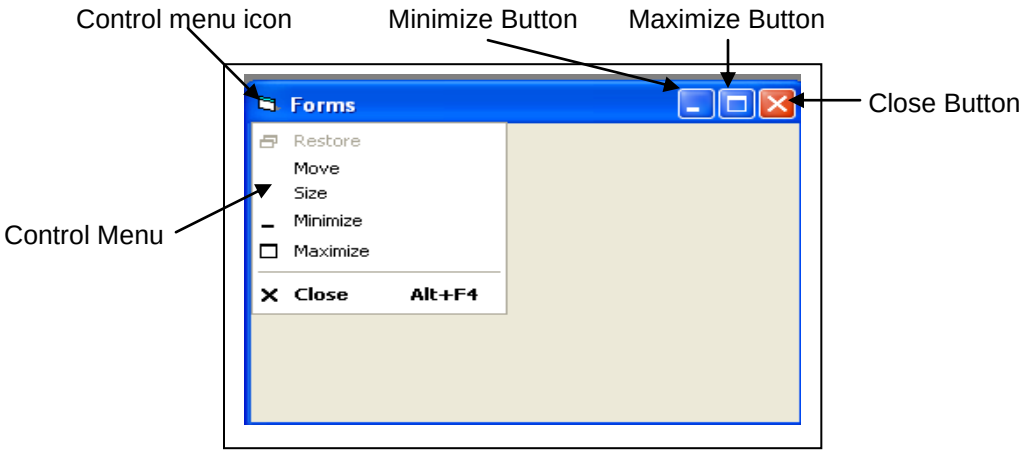
You assign the value 0 to the *num* variable before the loop starts because this value can't affect the total. The entered numbers will be printed on the form by statement 1. In case the user inputs a value less than 1, the loop will terminate.
It is possible to precede the 'While' statement with an 'InputBox' function to get the first number from the user.

WORKING WITH FORMS

In visual basic, the *Form* is the container for all the controls that make up the user interface. When the Visual Basic application is executing, each window it displays on the desktop is a Form. The Form is the top-level object in a Visual Basic application and every application starts with the Form. Forms have a built-in functionality that is always available without any programming effort on your part. You can move a form around, resize it and even cover it with other forms.

Appearance of Forms

The main characteristic of the Form is the title bar on which the Form's caption is displayed. On the left end of the title bar is the Control menu icon. Clicking this icon opens the control menu. On the right side of the title bar are three buttons: Minimize, Maximize and Close. Clicking this buttons performs associated functions. When a Form is maximized, the maximizing button is replaced by the Normal button. When clicked, the Normal button will restore the Form to its size and position before it was maximized.



The control menu contains the following commands:

- **Restore** -Restores a maximized Form to the size before it was maximized. It is available only if the Form has been maximized.
- **Move** -Lets the user move the Form around with the mouse.
- **Size** -Lets the user resize the control with the mouse
- **Minimize** -Minimizes the Form
- **Maximize** -Maximizes the Form
- **Close** -Closes the Form

You can customize the appearance of the Form with the following Form properties.

- **MinButton, MaxButton** -These two properties are 'True' by default. Set them to 'False' to hide the corresponding buttons on the title bar.
- **ControlBox** -This property is also 'True' by default. Set it to False to hide the icon and disable the control menu. Although the control menu is rarely used, Windows applications don't disable it. When the ControlBox property is False, the three buttons on the title bar are also disabled. If you set the Caption property to an empty string, the title bar disappears altogether.

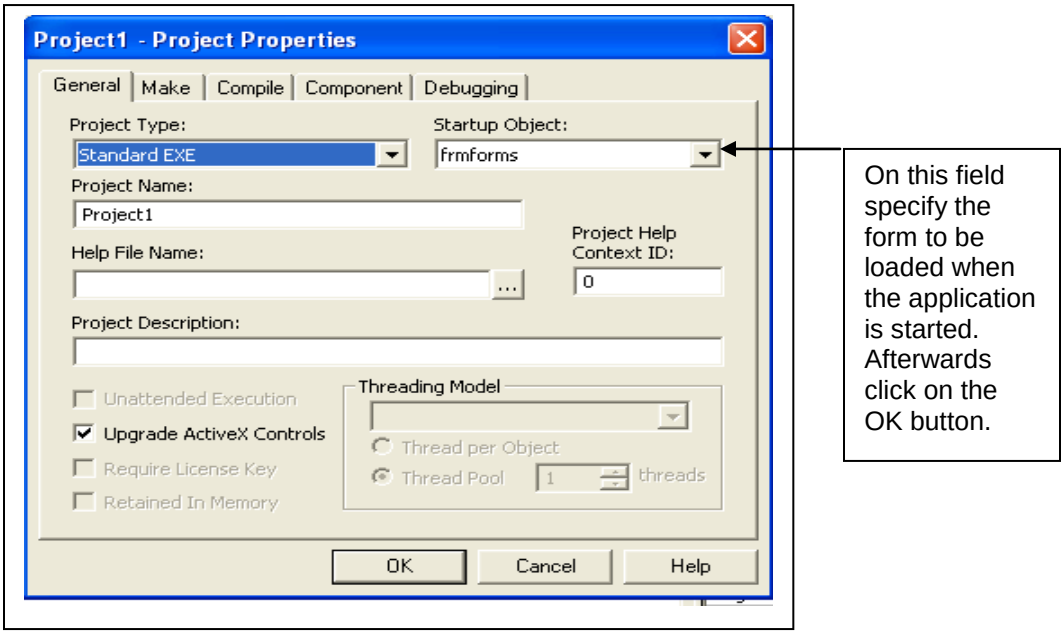
- **BorderStyle** -This property determines the style of the Form's bolder and the appearance of the Form. The BolderStyle property can take one of the values shown below.

Values of BolderStyle property

Value	**Constant**	**Description**
0-None	vbBSNone	Form has no bolder and can't be resized. This setting should be avoided.
1-Fixed Singe	vbFixedSingle	Form has a visible bolder, can't be resized
2-Sizable	vbSizable	Bolder and a title bar and can be repositioned on the Desktop and resized.
3-Fixed Dialog	vbFixedDialog	Fixed dialog box
4-Fixed ToolWindow	vbFixedToolWindow	Form has a close button only and can't be resized; looks like a toolbar
5- Sizable ToolWindow	vbSizableToolWindow	Same as the Fixed ToolWindow, but can be resized.

Start-Up Form

A typical application has more than a single Form. When the application starts, the main Form is loaded. You can control which Form is initially loaded by setting the start-up object in the Project Properties window shown below. To open this properties window, choose Project > Project Properties



Loading, Showing and Hiding Forms

The form has three possible states:

- **Not Loaded** -The form lives on a disk file and doesn't take up any resources.
- **Loaded but not shown** -The form is loaded into the memory, takes up the required resources and is ready to be displayed.
- **Loaded and shown** -The form is shown and the user can interact with it.

Loading and Unloading Forms

To load and unload Forms, use the 'Load' and 'Unload' statements. The following are the syntaxes for the two statements

Load *formName*

Unload *formName*

Where *formName* is the name of the form to be loaded or unloaded. Unlike the show method which takes care of both loading and displaying the Form, the Load statement doesn't show the Form. You have to call the Form's Show method to display it on the Desktop.

Once a form is loaded, it takes over the required resources, so you should always unload a form that's no longer needed. When a form is loaded, the resources it occupies are returned to the system and can be used by other Forms and / or applications. Because loading a form isn't instantaneous, especially if the form contains bitmaps or other resource that entail loading large files, don't unload a form frequently in the course of an application. If your application contains many forms, balance the benefits of having them all in memory versus loading certain form as needed. Opening a database and setting up the related structures, for instance, doesn't take place instantly, and your application shouldn't load and unload forms that access database. Its best to keep the form loaded in the memory and display it when the user needs it.

Showing Forms

To show a form, you use the show method. If the form is loaded but invisible, the show method brings the specified form on top of every other window on the Desktop. If the form isn't loaded, the show method loads it and then displays it. The show method has the following syntax:

formName.Show

The formName variable is the Form's name.

Hiding Forms

If your application uses many Forms, you may want to hide some of them to make room on the Desktop for others. To hide a Form, use the Form's Hide method, whose syntax is:

Form.Hide

To hide a form from within its own code, use the statement:

Me.Hide

Forms that are hidden are not unloaded; they remain in the memory and can be displayed instantly with the show method. Forms that may be opened frequently, such as a Search & Replace window, should be hidden when they are not needed. The next time the user opens the same window there will be no noticeable delay. Form's that aren't opened frequently in the course of an application, such as a Preferences or Option window, should be unloaded to reclaim the resources.

When a form is hidden, you can still access its properties and code. For example, you can change the setting of its control properties or call any public functions in the form, even its event handlers (as long as you make them public by changing the default "Private" declaration of the corresponding event handler to "public").

Designing Menus

Menus are one of the most common and characteristic elements of the windows user interface. Even in the old days of character-based displays, menus were used to display methodically organized choices and guide the user through an application and the many alternatives, menus are still the most popular means of organizing a large number of options. Many applications duplicate some or all of their menus in the form of icons on a toolbar, but the menu is a standard fixture of a form. You can turn the toolbars on and off, but not the menus.

The Menu Editor.

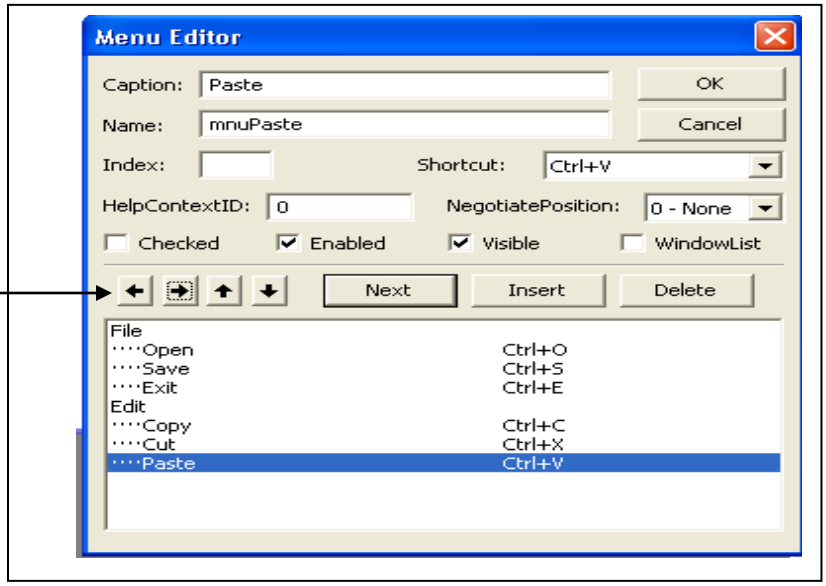
Menus can be attached only to forms, and you design them with the *Menu Editor*. To see how the Menu Editor works, start a new standard EXE project, and when Form1 appears in the design window, choose **Tool** > **Menu Editor** to open the Menu Editor, as shown below. Alternatively, you can click the menu Editor button on the toolbar.

In the Menu Editor window you can specify the structure of your menu by adding one command at a time. Each menu command has two mandatory properties:

- **Caption** - This is the string that appears on the application's menu bar.
- **Name** – this is the name of the menu command. This property doesn't appear on the screen, but your code uses it to program the menu command.

The *Caption* and *Name* properties of a menu item are analogous to the properties that have the same name as the Command button or label control. Caption is what the user sees on the form, and Name is the means of accessing the control from within the code. As far as your code is concerned, each menu command is a separate object, just like a command button or label control.

To add command to the Form's menu bar, enter a caption and a name for each command. As soon as you start typing the command's caption, it also appears in a new line in the list at the bottom of the menu Editor window. Let's create the menu structure shown below.



Caption and Name properties for File and Edit commands are shown in the table below.

CAPTION	NAME

Prepared by:
Ericobanks
Contact: 254700711233
Website: www.codestar.co.ke

File	mnuFile
Open	mnuOpen
Save	mnuSave
Exit	mnuExit
Edit	mnuEdit
Copy	mnuCopy
Cut	mnuCut
Paste	mnuPaste

The commands that belong to each menu form the corresponding submenu and are indented from the left. To design the menu, follow these steps:

1. Open a new Form in the Design pane, and choose Tools > Menu Editor to open the menu editor window.
2. In the Caption box, type the Caption of the first command (file).
3. In the Name box, enter the command's name (mnuFile).
4. Press Enter or click Next to enter the next command. Repeat step 1 through 3 for all the commands listed.
5. For the indented commands, use the arrow facing to the right to indent. When a command is indented, it's made to be inside the command on the higher level. The levels of the commands depend on the vertical arrangement of the commands. In the example given above, 'File' and 'Edit' are in the same level (highest level) while 'Open', 'Save' and 'Edit' are inside the 'File' command. The three (Open, Save and Edit) form the File menu. In the same way, 'Copy', 'Cut' and 'Paste' are inside 'Edit' command forming the Edit menu.

The Left and Right arrows are used to change the level of the commands. Left arrow moves the command to a higher level while right arrow to a lower level. The Up and Down arrows arrange the commands.

The Index Field

Use Index field to create an array of menu commands. All the commands of the array have same name and a unique index that distinguishes them, just like arrays of controls. When appending a list of recently opened filenames in the file menu, it's customary create an array of menu commands (one for each filename); all have the same name and a different index.

The checked field

Some menu commands act as toggles, and they are usually checked to indicate that they are on or unchecked to indicate that they are off. To display a checkmark next to a menu command initially, select the command from the list by clicking its name, and then check the checked box in the Menu Editor window. You can also access this property from within your code to change the checked status of a menu command at runtime. For example, to toggle the status of a menu command called Frmtbold, use the statement:

```
frmtBold.checked = Not frmtBold. Checked
```

The visible field

To remove a command temporarily from the menu, set the command's visible property to false. The visible property isn't used frequently in menu design. In general, you should prefer to disable

a command to indicate that it can't be used (some other action is required to enable it). Making a command invisible frustrates users, who try to locate the command in another menu.

The window field

This option is used with MDI (Multiple Document Interface) applications to maintain a list of all open windows.

Programming Menu commands

Menu commands are similar to controls. They have certain properties that you can manipulate from within your code, and they recognize a single event, the *click* event. If you select a menu command at design time, Visual Basics open the code for the click event in the code window. The name of the event handler for the click event is composed of the command's name followed by an underscore character and the event's name, as with all other controls. Type the codes on the coding section which will be executed when the menu will be clicked.

Access keys

Access keys allow the user to open a menu by pressing the Alt key and letter key. To open the Edit menu in all window application, for example, you can press Alt+ E. E is the Edit menu's access key.

DATABASE PROGRAMING

Relational Database concepts

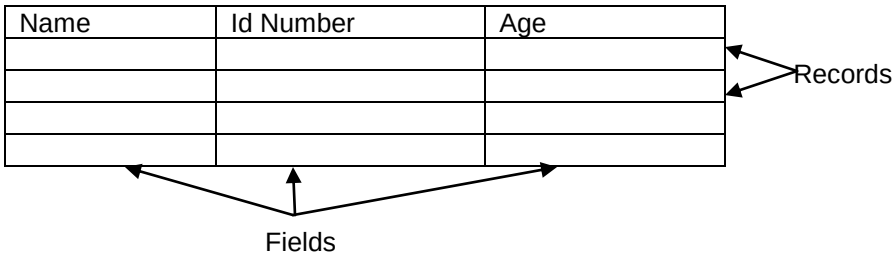
Database

This is an organized collection of related data about an organization. A database structure is made up of a number of components; the common being tables, queries and indexes. Therefore the database must consist of these elements in order to store data.

Table

It's a logical grouping of related information. A relational database presents data inform of a table. A table consists of rows referred to as *records* and columns referred to as *fields.*

Example:



Records -Each *record* in a table contains *information about a single entry* in a table e.g. in an employees table, a record will contain all the details for a particular employee.
Fields -Each *field* contains a *single piece of information* about a *record*. Several fields will constitute a record. E.g. Age , Id number etc.

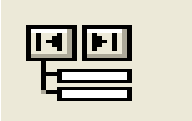
Primary key

- To identify each row in a table, then a primary key is required for the table. Primary key is a unique field or a combination of fields whose values are unique for each row. A table may also contain a field that is a *foreign key*. A foreign key points to a primary key in a related table. The foreign key should be the same as the primary key for the other table. It is required for creating relationship or links between the tables.

Creating Database in Visual Basic

When creating a database in visual basic, we use *Ms-Access* of *Visual Data manager*, which is an integrated part of Visual Basic. A table is then created inside the database to store the data.
When creating a table, the first thing is to identify the fields from the program form and their data types. This is because the fields on the form will be connected to the table.

Visual Data Control

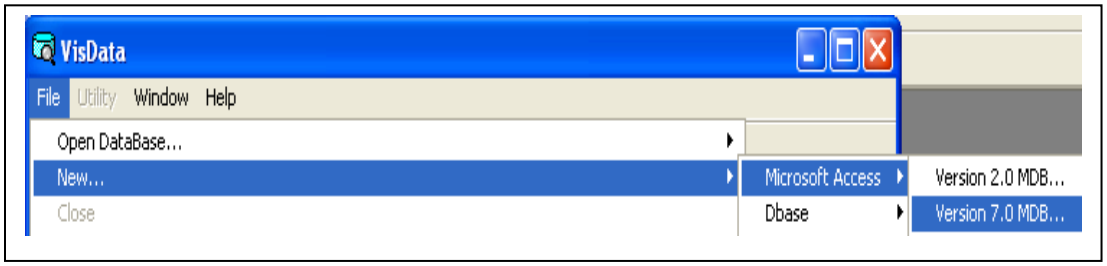


It's an access data object that enables the other controls in the form to be connected to the database. This control allows easy access to the database when adding, searching, deleting and updating records.
When a data control is placed on a form, it is connected to the database by use of *DatabaseName* property. To connect to the specific field in the table, the *RecordSource* property of the *Data* control is set to the specific table (table name).
For a control to be connected to the database using the *Data* control, the *DataSource* property of the control has to be set to the name of the *Data* control connected to the database.
To connect the fields in the table to the controls, the *DataField* property of the control has to be set to the specific field of the table.

Creating a Database

Steps:

i) From "Add-Ins" menu choose "Visual Data manager…".
ii) Make sure that "Table Type Recordset" and "Used Data control" are clicked.
iii) On Visual Data window, choose "New Ms Access version 7.0" from the "File" menu.
iv) Specify where to save the database.
v) Type the database name then click "Save".



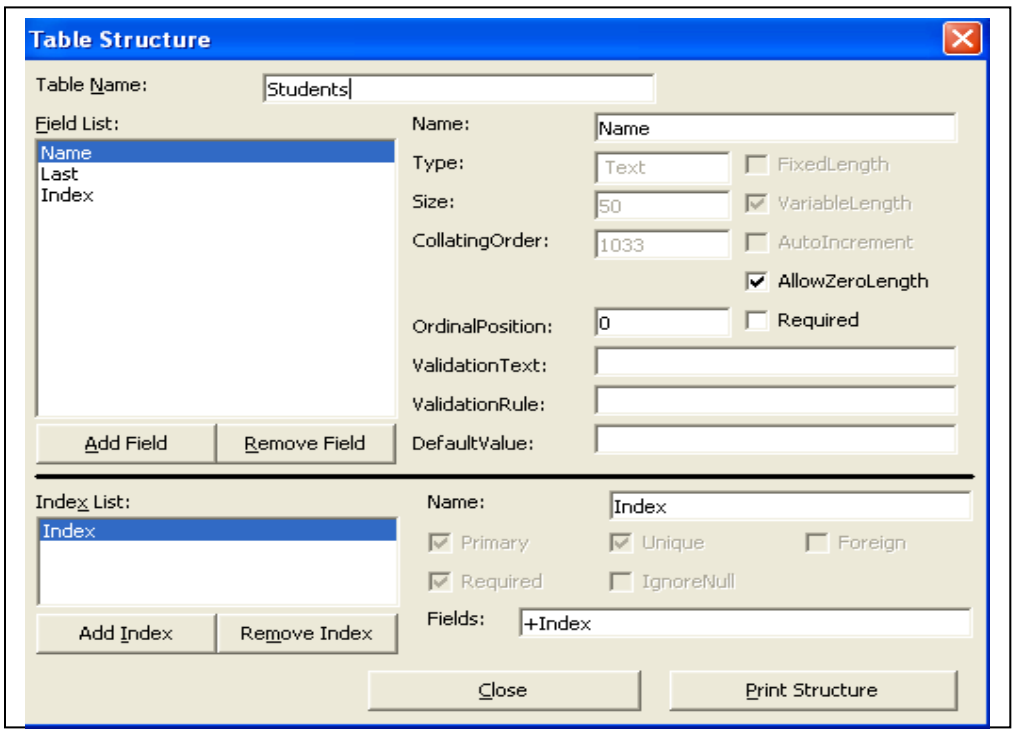
Creating a Table

The table is created inside the database window.

Steps:

i) While creating the database (after saving), on the window that appears, right-click on the window and click on "New Table". A "Table structure" dialog box appears.
ii) Type the name of the table.
iii) Click on the "Add Field" button to add the fields of the table that should correspond to the controls you have on the form.
iv) Type the name of the fields and select the data types one after the other by clicking the OK button always after entering the field.
v) After entering all the fields, click on the "Close" button.
vi) Click on the "Add Index" button to add the primary field. (This is the unique field in the table).
vii) On the "Available Fields" section, click on the field name that shows the primary field to be copied to the "Index Fields" section. Type the same name copied on the "Index Fields" on the "Name" field.
viii) Make sure that the "primary" and "unique" checkboxes are checked.

ix) Click on the OK button. If you need other fields to be unique and primary, then follow the same procedure as in step 7 and 8.
x) After adding all the indexed fields, click on close button.
xi) On the "Table structure" dialog box, click on "Build the table" to generate the table in the database.
xii) Close the "Visdata" window to go back to the form.



Connecting the Database to the form controls

At this point, the form controls are not connected with the database and as a result the program won't save the entered data.

The following are the steps to follow in order to connect the controls with the database. Make sure that there is a "Data" control placed in your form because this is the control used to connect the database to the controls on the form.

Steps:

i) If the "data" control is not there, place it on the form and give it a name. write the caption of the "Data" control also.

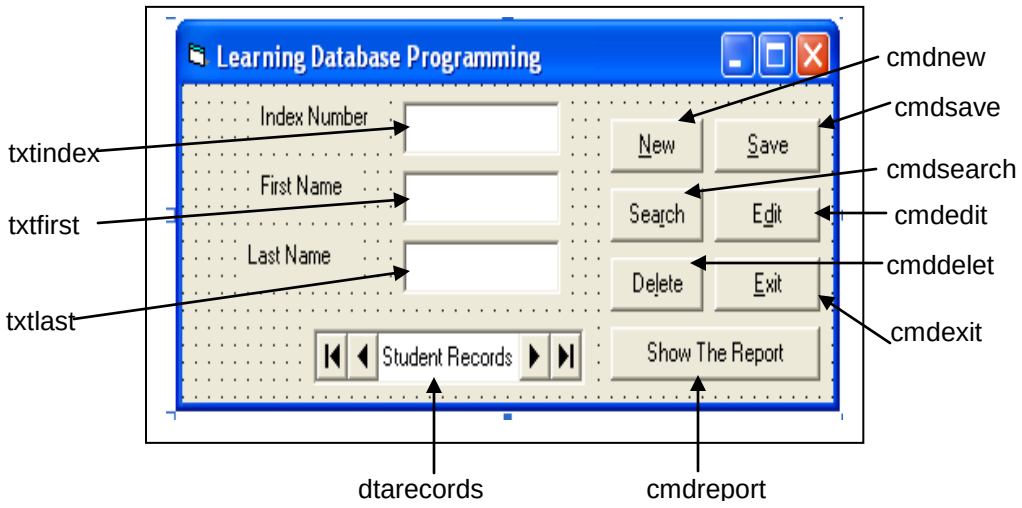
ii) Open the properties window for the "Data" control and connect the "Data" control to the database through the "Databasename" property. Click on the button on this property to and click on "open" to connect the "Data" control to the database.
iii) Still on the properties of the "Data" control, connect "Recordsource" property to the table you created on the database. A database can have several tables, so connect the table that you want to use to be storing the data entered through the controls you have in your form. At this point, the "Data" control as connected to the Database and the table. The remaining bit is to connect the other controls to the database.
iv) Since the other controls are to be connected to the database; this is done through connecting them to the "Data" control. Click on the first control to be connected to the database and open its properties window. On the "Datasource" property, click on the drop-down arrow and click on the name of the "Data" control connecting to the database. On the "Datafield" property, click on the drop-down arrow and select the field (from the table you connected to the "Data" control) that corresponds to the control you are connecting to the database. Repeat the same procedure of connecting the "Datasource" and "Datafield" to all the other controls to be connected to the database. NB; Before connecting the "Datafield" property of any control, first connect the "Datasource" property. This will let the table fields to be displayed on the "Datafield" property for the programmer to select.
v) Remember to click on the save tool on the tool bar or choose "save" from "file" menu to save the connections.

At this point the controls on the form are connected to the database but there is nothing that the program can save to the database because there are no codes put on the command buttons.

Coding
To understand very well on how to code, we need to come up with an example which is working.

Example 20:
Design a program that will save student names and their index numbers in a database.

Step 1: Design the GUI as shown below.



Step 2: Coding

1) Creating the Database and a table

Following the steps on how to create a database as explained on the previous page, create the database and give it the name *"Studentrecords".*
Create a table in the database following the steps given on the previous page and give it the name *"Students".*
Add "Index", "First" and "Last" as the data field on the table with "text" as the datatype. Add "Index" datafield as the index field of the table. This should be *primary* and *unique*. This field will be used during searching of the data in the database.

2) Connecting the database with the form controls

Open the properties window of the "Data" control *dtarecords* and set the following properties.

i.) DatabaseName – Use the browse option to connect it to the database *"Studentrecords"*

ii.) Recodsource - Click the drop down arrow and choose the table *"Students"*

For each of the text boxes set the "Datasource" to *"dtarecords"* by use of the down arrow. Set the appropriate "Datafield" for each of them using the down arrow as indicated below.

Field	Datasource	Datafield
txtindex	dtarecords	Index
txtfirst	dtarecords	First
txtlast	dtarecords	Last

The "Datasource" for each field should be set fist and then the "Datafiled" follows.

3) Necessary codes for the six command buttons

i) New command button

```
Private Sub cmdnew_Click()
dtarecords.Recordset.AddNew ……1
cmdsave.Enabled = True
cmdsearch.Enabled = True
cmdedit.Enabled = True
cmddelete.Enabled = True ……2
txtindex.Enabled = True
txtfirst.Enabled = True
txtlast.Enabled = True
cmdnew.Enabled = False ……3
txtindex.SetFocus ……4
End Sub
```

ii) Save command button

```
Private Sub cmdsave_Click()
If txtindex.Text = "" Then ……5
MsgBox ("You can not save a Null Entry.") ……6
txtindex.SetFocus ……7
Else
dtarecords.Recordset.Update ……8
cmdnew.Enabled = True ……9
cmdsave.Enabled = False ……10
End If
End Sub
```

iii) Search command button

```
Private Sub cmdsearch_Click()
Dim sindex As String ……..11
Dim sbookmark As String ……12
sindex = InputBox("Type the student index number") ……13
With dtarecords.Recordset ……14
sbookmark = .Bookmark ……15
.FindFirst "Index like ' " + sindex + " ' " ……..16
If .NoMatch Then ……..17
MsgBox ("Student does not exist") ……..18
.Bookmark = sbookmark ……..19
Else
cmdedit.Enabled = True ……20
cmddelete.Enabled = True ……21
End If
End With
End Sub
```

iv) Edit command button

```
Private Sub cmdedit_Click()
dtarecords.Recordset.Edit ……22
txtindex.Enabled = True
txtfirst.Enabled = True } ……23
txtlast.Enabled = True
cmdsave.Enabled = True ……24
End Sub
```

v) Delete command button

```
Private Sub cmddelete_Click()
With dtarecords.Recordset ……..25
If MsgBox("Are you sure that you want to delete this record?", vbYesNo) = vbYes Then
MsgBox ("The record will be permanently deleted.") ……..27
.Delete ……..28
.MoveFirst ……..29
If .EOF Then ……30
.MoveLast ……31
End If
End If
End With
End Sub
```

vi) Exit command button

```
Private Sub cmdexit_Click()
Unload Me ……32
End Sub
```

vii) Form load code

```
Private Sub Form_Load()
txtindex.Enabled = False
txtfirst.Enabled = False } ……33
txtlast.Enabled = False
```

cmdsave.Enabled = False ……..……..34
frmdatabase.Top = 2000
frmdatabase.Left = 2500 } ……35
End Sub

Explanation:

- **Form load**

The form-load event is executed when the form loads. This is when the program is made to run.
The controls (text boxes) indicated are disabled (statements 33) because the user may start typing the data straight away without first clicking on the "New" button. The "New" button is clicked first before entering the data to prepare the database to receive the entered data. This is done by the code on statement 1. If the data will be typed without first clicking on the "New" button, the action will be cancelled by the program.
The "Save" button is disabled (statement 34), to avoid the user clicking on it before first entering the data for this will bring an error of saving without first clicking on "New" or "Edit" buttons. When these two buttons are clicked they prepare the database to receive data.
Statements 35 are made to position the form at a particular point depending on the dimensions set by the programmer. This is not a must to be put but for more professional looking program they are important.

- **New command button**

The code on statement 1 is used to prepare the database to receive the data entered through the form. As the programmer, you should make sure that before the user types and saves any data, has clicked a command button with this code.
After clicking on this command button, the other command buttons together with the textboxes will be enabled (statements 2). At the same time, this command button will be disabled to avoid the user from clicking on it again (statement 3).

- **Save command button**

Remember that the field for the index is primary and unique. It is also a must to put some data in this field. Because of this characteristic of the field, statement 5 is made available on the code to avoid having an error if the user clicks on the "Save" button before entering the index. The double quotes on statement 5 signify that there is no data entered in the text box *txtindex*. As a result, the user will get a message (statement 6) and statement 7 will take the curser to the textbox *txtindex* for a value to be entered. The user can enter the index alone and save and get no error. This because it is the only required entry in this form. A program can have several fields which require data before saving the data in the database. No saving will take place if the fields will be containing nothing when the user clicks on the save button.
If the index is supplied, then by clicking on the save button, statement 8 will be executed. This will save the entered data in the database.
This button will enable the "New" button (statement 9) for more entries and disable itself (statement 10) because if the user will click on it again before clicking on the "New" button then it will lead to an error.

- **Search command button**

The codes will help the user to search or an entry in the database. Statements 11 and 12 are used to declare the variable names *"sindex"* and *"sbookmark".* Statement 13 is used to assign the value entered through the input box generated by the function *"Inputbox( )",* to the variable name *"sindex".*
Statement 14 will direct the computer to the database to search the requested entry. The database to be searched is the one connected to the data control *"dtarecords".*
Statement 15 sets the "Bookmark" which is used to identify that particular entry during the search process. Statement 16 finds the first entry in the table with the field "Index" that resemble the

value assigned to the variable name *"sindex".* The word "Index" should be the name given to the unique and primary field in the database.
Statement 17 is executed if there is no entry matching with the entered value. Therefore a message box will be generated (statement 18). After the message box, statement 19 is executed which resets the searching since there is no matching entry in the table.
If there is a matching entry, then the statements after the "Else" keyword (20 and 21) will be executed and the matching entry will be displayed on the form controls. The "Edit" and "Delete" buttons are enabled because the user may decide to edit or delete the record.

- **Edit command button**

Statement 22 on the code prepares the database for some editing (changes) and even makes it ready to save the changes made. The code will also enable the text boxes (statements 23) for editing and the Save button (statement 24) for saving the change made. Incase the user tries to edit without clicking the edit command then, an error will occur during the saving process.

- **Delete command button**

Statement 25 prepares the database to delete the desired record (displayed record on the form controls) from the records.
Statement 26 generates a message in a message box with two button "Yes" and "No". If the user clicks on "Yes" statement 27 will generate a message box to warn the user on the effect of deleting the record. Statement 28 will delete the entry displayed on the form controls and 29 will display the first entry in the database on the form controls.
The "EOF" on statement 30 means "End Of File" and this will be executed incase the user deletes the last entry in the database. In such a case, statement 31 will be executed which will display the last entry in the database.

- **Exit command button**

The command on this button is used to stop the running of the program. This ends the program. It will act the same way as "End" command.

Assignment 6;
Modify the program such that it will not be deleting the displayed entry on the form once the delete command button is clicked but it will be requesting the user to input the index of the entry to be deleted. The program will be searching for the entry to be deleted and after finding it, it will be requesting if to delete or not. Incase the user chooses to delete the entry will be deleted otherwise the entry will not be deleted.
Incase the program fails to find the index of the entry to be deleted; it should inform the user and give him/her another chance to re-enter the index again.
Also, Modify the codes on the search button such that incase the index being searched is not found, it will be notifying the user and give him/her another chance to re-enter the index number for a new search.

Generating Reports

Reports are generated from tables and show the summary of the data from that particular table. They are used for management purpose such as analysis.
The most common control used in generation of reports is the *DataEnvironment.* This object is flexible is that it can be used in connecting controls placed on the form to the database. For a *Data* control to be connected to the database a *Connection* must be established. This specifies the path of the database and the database provider used in the connection. To get connected to a table, a *Command* is created.
In general, generating a report follows the given steps;

1. Adding Data Environment.
2. Establishing a Connection
3. Adding a Command
4. Creating the Data report.

We will create a report for the above example.

- **Data Environment**

This is the first step in creating the data report. It is designed within visual Basic and is used to tell the report what is in the database.

Steps to Add Data Environment

i.) From "Project" menu click on "Add Data Environment". Data Environment window appears.
ii.) Click on "DataEnviroment1" and open its properties window. Enter the name property as *"DataEnvstudents".*
iii.) Click on "connection1" and open its properties window. Set the name as *"connstudents".*
iv.) Right click on the connection *"connstudents"* and click on "Add command".
v.) Click on the command added and open its properties window. Set the name as *"commstudents".* Make sure that the property "ConnectionName" has the name of the connection, *"connstudents".*
vi.) Right click on the connection and choose "Properties". On the "Data link properties" dialog box that appears, click on the "Provider" tab and click on "Microsoft jet 3.51 OLE DB provider". Click on the "Next" button on the "Connection" tab.
vii.) Enter the database name by clicking on the browser button next to *"select or enter a database name:"* field and navigating through until you open the required database. In our case it should be *"studentrecords".* Click on "Open" button to connect the connection with the database.
viii.) Click on the "Test connection" button to confirm the connection. On success a message box with the message, "Test connection succeeded" is displayed. Click on the OK button and OK again to go back to the "Data Environment" window.
ix.) Right click on the command *"commstudents"* and choose "Properties". A dialog box appears.
x.) Click on the "General" tab and confirm that on the *"Command Name"* and *"Connection"* fields have the names of the Command and the Connection, *"commstudents"* and *"connstudents"*, respectively.
xi.) On the "Database object" field, click on the down arrow and choose "Table".
xii.) Click on the SQL statement option button and type the following.

```
SELECT Students.* FROM Students
```

Where:

Students - is the name of the table in our database.
Student.* - is used to retrieve all the fields from the table "Students". Incase one is interested to retrieve particular fields from the table, then instead of putting an asterisk after the period on table name, should specify the field names as used in the table and separate each set of table name and field name with a comma.
Example: If we want to retrieve only index and first name we will type the following.

```
SELECT Students.Index,Students.First FROM Students
```

Where: "Index" and "First" should be the field names on the table "Students".

- SELECT and FROM should be typed in upper case only

- There should be a single space between SELECT & table fields, table fields & FROM and FROM & the table name.
- On the success of the connection,a plus (+) sign will appear on the SQL command on the "Data Environment" window. When you click on the plus (+) sign, a list of all the retrieved fields from the table will be displayed.
- By opening the "Project Explorer" window, a folder named "Designers" will be found containing the data environment created.

At this point you are through with creating the Data environment.

- **Data Report**

This is the second stop in generating a data report. It is designed in the visual Basic window. Once the Data Report and Data Environment are designed they become part of visual Basic project.

Steps to add Data Report

i.) From "Project" menu click on "Add Data Report". A data report window will be displayed.

ii.) Open the "Properties" window of the Data Report and set the following properties.

Name – DataRepStudents
Caption – Student Records
Datasource – DataEnvstudents; this is the name of the Data Environment we created. Click on the down arrow to select it.
Datamember – Commstudents; this is the command name we created. Also use the down arrow to set it


Before setting the "Datamember" , the "Datasource" property has to be set first.

The Data Report window has five sections as follows:

c) **Report Header**- It contains the information displayed on top of the whole report. A report can have several pages, so the information placed in this section will be displayed on the first page on the report. Example: The name of the institution.

d) **Page Header**- It contains the information displayed on top of each page e.g. student exam records.

e) **Detail section** – This section contains the fields from the data environment. To add the fields, open Data Environment and drug the fields into this section.

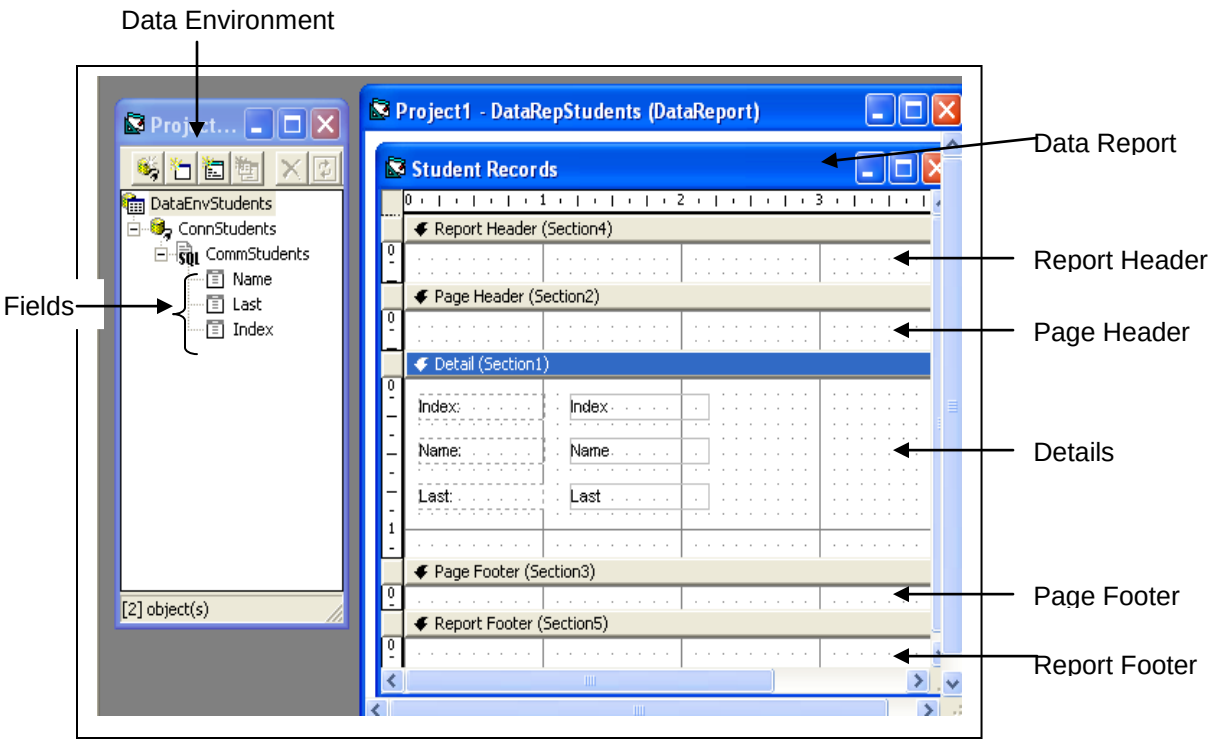
f) **Page footer** – It contains the information displayed at the bottom of each page on the report. e.g. Student department name.

g) **Report footer** – It contains the information displayed at the bottom of the whole report. This information appears on the last page of the report.

The details section can contain a single record from the Database or several depending on the amount of information on each record.

iii.) Open both the Report window and the Data Environment window. Make sure that, you resize them to a point that both will be seen on the same computer screen.

iv.) Click on the fields from the Data Environment and drag them into the Data Report details section. Position them where necessary to give the best output of the report.



Modifying the appearance of the report

The data report has its tool box that contains controls that can be used to modify the report. To put a label on the different sections as the headers and the footers, click on the "Rptlabel" control on the tool box and draw it where necessary on the report. This is done in a similar manner to the designing of the GUI of the program.

Each control has its properties so by opening the properties window of a particular control, will help you to format the controls.

Put the labels for the report header, page header, page footer and report footer and format them.

The fields dragged and dropped on the details section are in pairs. The first one (the one on left) is the filed label while the other (one on the right) is the field which will display different entries entered through the form. It is possible to modify the appearance of the fields on the details section by clicking on each of them and opening its properties window. On the properties window you can format as required by your report. For the labels dragged from the fields, you can change the caption because the caption displayed on the report by default is obtained from name given to the connected table on the database.

Viewing the Report through the form

On the form, a command button can be added which will be displaying the report once clicked. This is similar to the way we display other forms. Therefore the required code will have the name of that particular report.

For instance, if we want to have this report displayed on clicking on a command button we will add it on the form and code the following on the coding window.

```
Private sub cmdreport_click( )
DataRepStudents.Show
End sub
```